In compliance with the
Canadian Privacy Legislation
some supporting forms
may have been removed from
this dissertation.

While these forms may be included
in the document page count,
their removal does not represent
any loss of content from the dissertation.

University of Alberta

Computational Intelligence–based Classifiers in Multi-dimensional Pattern Recognition

by

Arnon Breuer

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Master of Science

Department of Electrical and Computer Engineering

Edmonton, Alberta
Fall 2003

National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisisitons et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 0-612-87725-6
Our file Notre référence
ISBN: 0-612-87725-6

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou aturement reproduits sans son autorisation.

Canada

University of Alberta

Library Release Form

Name of Author:	Arnon Breuer
Title of Thesis:	Computational Intelligence-based Classifiers in Multi-dimensional Pattern Recognition
Degree:	Master of Science
Year this Degree Granted:	2003

Permission is hereby granted to the University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves all the other publications and other rights in association with the copyright in the thesis, and except as herein before provided, neither the thesis not any substantial portion thereof may be printed or otherwise reproduced in any material form whatever without the author's prior written permission.

Date: July-4-03

University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled Computational Intelligence-based Classifiers in Multi-dimensional Pattern Recognition submitted by Arnon Breuer in partial fulfillment of the requirements for the degree of Master of Science.

Dr. Aminah Robinson

Date: July - 4 - 03

Abstract

Automatic pattern recognition is becoming overwhelmingly important in a variety of areas such as medical diagnosis, engineering and economics. This multitude of data inspires development of new and powerful computational tools.

Computational intelligence (CI) combines in synergy several advanced information processing technologies: neural networks, fuzzy sets theory and evolutionary computation. The objective of this study is to investigate three methods in which CI is involved with pattern recognition:

- A feature extraction method using a genetic algorithm (GA). The GA serves as an optimization tool for constructing a piecewise representation of highly dimensional data.
- A fuzzy adaptive logic network (FALN) used as a pattern classifier. The FALN realizes a topology of perceptrons combined with a structure of fuzzy neurons, which has the advantages of both processing capabilities and transparency.
- Lastly, FALNs are incorporated in an ensemble (using bagging and boosting methods) for achieving improved accuracy.

Dedicated to Orna, Ofer, Inbar and Talia

Acknowledgements

I would like first to convey my gratitude and appreciation to my supervisor, Dr. Witold Pedrycz for his constant guidance, his professional and supportive attitude and the numerous comments and suggestions he made throughout the preparation of this thesis.

I would also like to thank Dr. N. J. Pizzi from the Institute of Biodiagnostics, National Research Council (NRC), Canada, for providing the biomedical data used in this research.

Support from the Canada Research Chair (CRC) Program (W. Pedrycz), Natural Sciences and Engineering Research Council (NSERC), and Alberta Software Engineering Research Consortium (ASERC) is gratefully acknowledged.

I would like to thank Rafael, Israel for the opportunity to study here and for the financial assistance.

Finally, I would like to express my love and thanks to my family, Orna, Ofer, Inbar and Talia.

Table of Contents

1. Introduction 1
2. Computational intelligence and pattern recognition – an overview 6
2.1 Introduction 6
2.2 Phases in pattern recognition 6
2.2.1 Feature extraction 7
2.2.2 Classification 7
2.3 Computational intelligence building blocks 8
2.3.1 Neural networks 8
2.3.2 Fuzzy sets 9
2.3.3 Evolutionary computation 10
2.4 Computational intelligence for pattern recognition 12
2.4.1 Neural networks and fuzzy sets 12
2.4.2 Evolutionary computation and neural networks 13
2.4.3 Evolutionary computation and fuzzy sets 14
2.5 Summary 14
3. Data and methodology 15
3.1 Introduction 15
3.2 Data sets 15
3.2.1 Synthetic data 15
3.2.2 High dimensional biomedical data sets 18
3.2.3 Software engineering data 21
3.2.3 Machine learning repository data 22
3.3 Validation methodology 23
3.4 Summary 23
4. GA-based feature space generation 24
4.1 Introduction 24
4.2 Biomedical data – Candida species 24
4.3 Piecewise curve approximation as a model of a feature space: analysis of approximation capabilities 25
4.3.1 Background 25
4.3.2 Application to artificial data 33
4.3.3 Piecewise approximation of a single curve 37
4.3.4 Algorithm expansion: construction of an optimal piecewise approximation for the whole data set 46
4.4 Classification 50
4.4.1 Using the GA-based method extracted parameters 51
4.4.2 Using principal component analysis for feature extraction 53
4.5 Summary 58
5. Fuzzy adaptive logic networks 59
5.1 Introduction 59
5.2 Background 59

5.3 The network's architecture 60
5.3.1 Logic networks – some previous studies 62
5.4 Fuzzy neurons and fuzzy logic processing 64
5.5 Illustrative examples 69
5.6 Detailed learning scheme 71
5.7 Two design issues: structure selection and initialization of the learning 75
5.8 Experimental studies 82
5.9 Summary 109
6. Ensembles of FALNs 110
6.1 Introduction 110
6.2 Ensemble methods 110
6.2.1 Bagging 111
6.2.2 Boosting 112
6.3 Experimental studies 115
6.3.1 Medical Imaging System (MIS) data 115
6.3.2 High dimensional biomedical data 130
6.4 Summary 134
7. Conclusions and recommendations for future work 135
7.1 Conclusions 135
7.2 Future work 136
8. References 138

List of Tables

Table 1: Description of the MIS dataset with a detailed characterization of the software measures (adopted from Munson and Khoshgoftaar, 1996) 22
Table 2: Pseudo-code for the bottom-up segmentation algorithm 27
Table 3: Tuning of the GA parameters; approximation errors for various GA parameters; using two synthetic data curves and one biomedical spectrum 31
Table 4: Comparison of two GA approaches; approximation errors for two chromosome representations; using two examples of data curves 32
Table 5: Piecewise curve approximation: error magnitude average (± standard deviation); comparison of two piecewise linear methods; using artificial data 37
Table 6: Piecewise curve approximation: error magnitude average (± standard deviation); comparison of two piecewise linear methods, and one piecewise polynomial (order 2); biomedical spectral data 45
Table 7: Piecewise curve approximation of whole datasets: error average (± standard deviation); using 5, 10 ,20 and 30 segments; 100% vs. 10% of examples used per GA generation 48
Table 8: Average approximation errors; using all dataset, with 5, 10, 20 and 30 segments; 100% vs. 10% of examples used per GA generation 49
Table 9: Correct classification rates using the piecewise curve approximation parameters 52
Table 10: Correct classification rates for the four classification problems; using PCA with different numbers of PCs 58
Table 11: t-norm examples 65
Table 12: s-norm examples 65
Table 13: The detailed formulas for the update of the connections of the FALN 74
Table 14: Performance of FALN: (I- incremental learning; B- batch learning) 90
Table 15: The effect of initialization on learning 93
Table 16: Classification rates (± standard deviation): Infrared spectra of synovial joint fluid 105
Table 17: Classification rates (± standard deviation): Infrared spectra of synovial joint fluid with FALN determined and initialized using MSM-T 105
Table 18: Classification rates (± standard deviation): MR spectra of brain neoplasms 106
Table 19: Classification rates (± standard deviation): MR spectra of brain neoplasms with FALN determined and initialized using MSM-T 106
Table 20: Classification rates (± standard deviation): MR spectra of a biological fluid 107

Table 21: Classification rates (± standard deviation): MR spectra of a biological fluid with FALN determined and initialized using MSM-T 107
Table 22: Classification rates (± standard deviation): Wine data 108
Table 23: Classification rates (± standard deviation): Pima Indian diabetes data 109
Table 24: The Bagging algorithm pseudo-code 112
Table 25: Pseudo-code for the AdaBoost algorithm 114
Table 26: Classification problem MIS #1: classification rates (along with their standard deviation) 121
Table 27: Classification problem MIS #1: classification rates (± standard deviation) and their comparison with the ensemble methods 122
Table 28: Classification problem MIS #2: classification rates (along with their standard deviation) 123
Table 29: Classification problem MIS #2: classification rates (± standard deviation) and their comparison with the ensemble methods 124
Table 30: Classification problem MIS #3: classification rates (along with their standard deviation) 125
Table 31: Classification problem MIS #3: classification rates (± standard deviation) and their comparison with the ensemble methods 126
Table 32: Classification problem MIS #4: classification rates (along with their standard deviation) 127
Table 33: Classification problem MIS #4: classification rates (± standard deviation) and their comparison with the ensemble methods 128
Table 34: IR spectra of synovial fluid: classification rates (± standard deviation) and their comparison with the ensemble methods 131
Table 35: MR spectra of brain neoplasms: classification rates (± standard deviation) and their comparison with the ensemble methods 132
Table 36: A biological fluid: classification rates (± standard deviation) and their comparison with the ensemble methods 133

List of Figures

Figure 1: Thesis roadmap of empirical analysis 5
Figure 2: Pattern classification phases 6
Figure 3: Feed-forward neural network classifier with one hidden layer 9
Figure 4: Example of partitioning to four triangular membership functions 10
Figure 5: Main concept of genetic algorithm 11
Figure 6: Computational intelligence: synergy of neural networks, fuzzy sets and evolutionary computation 12
Figure 7: A two-class spiral dataset (N=96) 16
Figure 8: Non-linear data sets: piecewise linearly separable data (a), clustered data (b), highly structured data (c) 17
Figure 9: Plots of typical samples of two classes from each of the biomedical data problem 19
Figure 10: Examples of biomedical data: IR spectra of synovial fluid 20
Figure 11: MR spectra of brain neoplasms 20
Figure 12: MR spectra of a biological fluid 21
Figure 13: Nine approximations using bottom-up segmentation with different number of segments; illustrated on AL data spectrum example 28
Figure 14: Nine approximations using GA-based optimization with different number of segments; illustrated on AL data spectrum example 33
Figure 15: Data and approximation of piecewise linear function using 8 linear segments 34
Figure 16: Data and approximation of a smooth function using 10 linear segments 35
Figure 17: Data and approximation of a smooth function with noise (noise standard deviation = 0.5) using 10 linear segments 36
Figure 18: Fitness values (maximum and average) as a function of generation number; spectrum no. 1 from class AL; using 20 segments 38
Figure 19: Data and approximation of AL example no. 1 using GA-optimized method, piecewise polynomial (order 2) 39
Figure 20: Data and approximation of AL example no. 1 using 20 linear segments 40
Figure 21: Data and approximation of GL example no. 1 using 20 linear segments 41
Figure 22: Data and approximation of KR example no. 1 using 20 linear segments 42
Figure 23: Data and approximation of PA example no. 1 using 20 linear segments 43
Figure 24: Data and approximation of TR example no. 1 using 20 linear segments 44

Figure 25: Piecewise linear approximation average errors, of the five data class using two approximation methods with 5,10, 20 and 30 segments 46
Figure 26: Distribution of breaking points 50
Figure 27: First two principal components; AL vs. GL dataset 54
Figure 28: First two principal components; AL vs. KR dataset 55
Figure 29: First two principal components; AL vs. PA dataset 56
Figure 30: First two principal components; AL vs. TR dataset 57
Figure 31: The geometry of perceptrons and their combinations along with their logic interpretations 60
Figure 32: Fuzzy adaptive logic network: a general topology 61
Figure 33: A generic topology of an ALN 63
Figure 34: A piecewise linear boundary and its ALN representation 64
Figure 35: Examples of t-norms 66
Figure 36: Examples of s-norms 67
Figure 37: Characteristics of OR neuron; with $w_1 = 0.8$, $w_2 = 0.3$ 68
Figure 38: Characteristics of AND neuron; with $w_1 = 0.6$, $w_2 = 0.1$ 69
Figure 39: 3D plots of the characteristics of the three perceptrons used in the network 70
Figure 40: 3D and contour plots of the fuzzy neural network for selected combinations of the connections of the AND neuron 71
Figure 41: The detailed topology of the network; shown are all connections and sizes of the layers 73
Figure 42: Linear classification tree example 80
Figure 43: Network equivalent to the linear classification tree 81
Figure 44: FALN equivalent to the linear classification tree 82
Figure 45: 3D and contour plot of the fuzzy adaptive network 83
Figure 46: Two-dimensional two-class data 83
Figure 47: 3D characteristics of the AND neuron and the network 85
Figure 48: Difference between the characteristics of the AND neuron and the network 85
Figure 49: A two-class spiral dataset 86
Figure 50: Performance index (RMSE) in successive learning epochs 86
Figure 51: Plots of the decision boundaries formed by the FALN (contour and 3D plot) 87
Figure 52: Example of random initialization of learning 88
Figure 53: Example of pseudo-inverse initialization of learning 89
Figure 54: Noisy dataset for evaluation of generalization capabilities of the network 89
Figure 55: Piecewise linearly separable two-dimensional data 91
Figure 56: Piecewise linearly separable two-dimensional data; pseudo-inverse initialization 92

Figure 57: Piecewise linearly separable two-dimensional data; MSM-T initialization 92
Figure 58: Experiment 4; synthetic data – two-class problem and classification boundaries 94
Figure 59: Two-class data and the performance of the FALN (2,1):
classification decision and 3D plot of the classification characteristics 95
Figure 60: Classification with FALN(4,3): classification boundaries
and 3D characteristics of the classifier 97
Figure 61: Experiments with FALN(5,3)
with their decision boundaries and 3D characteristics 98
Figure 62: FALN(8,4) with their decision boundaries
and 3D characteristics 99
Figure 63: Consistency measure with respect to window length for averaging
IR spectra of synovial fluid 101
Figure 64: Consistency measure with respect to window length for averaging
MR spectra of brain neoplasms 101
Figure 65: A classifier ensemble of neural networks 111
Figure 66: Values of connections for eight different FALNs being used in the ensemble
formed by the bagging method 118
Figure 67: Boosting method: values of the connections of five FALNs
along with the distribution of the weights attached to the individual data points 119
Figure 68: "Difficult" (to classify) data point in the feature space 120
Figure 69: "Difficult" (to classify) data point with the superimposed
data points coming from the testing set 120
Figure 70: Classification rates for the ensemble of FALNs; FALN(3,2) for the MIS
classification problem #1 treated as a function of ensemble size: bagging and boosting 122
Figure 71: Classification rates for the ensemble of FALNs; FALN(2,1) for the MIS
classification problem #2 treated as a function of ensemble size: bagging and boosting 124
Figure 72: Classification rates for the ensemble of FALNs; FALN(3,2) for the MIS
classification problem #3 treated as a function of ensemble size: bagging and boosting 126
Figure 73: Classification rates for the ensemble of FALNs; FALN(3,2) for the MIS
classification problem #4 treated as a function of ensemble size: bagging and boosting 128
Figure 74: Averaged error rates results for all networks and all MIS classification problems:
bagging and boosting 129
Figure 75: Classification rates for the ensemble of FALNs; FALN(3,1) for IR spectra
of synovial fluid treated as a function of ensemble size: bagging and boosting 131
Figure 76: Classification rates for the ensemble of FALNs; FALN(3,1) for MR spectra
of brain neoplasms treated as a function of ensemble size: bagging and boosting 132

Figure 77: Classification rates for the ensemble of FALNs; FALN(2,1) for MR spectra of a biological fluid treated as a function of ensemble size: bagging and boosting 133
Figure 78: A heterogeneous neural structure with the logic layer interacting with the numeric interface constructed by perceptrons and RBFs 137

1. Introduction

Pattern classification is an important and well-defined task within pattern recognition (Bishop, 1995; Duda *et al.*, 2001), which involves determining to which class belongs a certain collection of data points (such as an image or a segment of speech). Such a set of data points is commonly termed *feature vector*. The task of pattern classification can be seen as consisting of two main functional phases: the *feature extractor* and the *classifier*. High dimensional pattern classification is often especially difficult mainly due to the following:

- In order to arrive at a statistically meaningful analysis a large number of examples should be available and in many applications data is scarce.
- The function associating data-to-class can be extremely complex and highly non-linear, in particular with high dimensional data, and the solution can be very intricate.
- The search for a solution can lead to implementation difficulties such as: long execution times (when using iterative methods), large memory requirements and numerical instabilities.
- Frequently data may be noisy or incomplete (missing features), leading to additional difficulties in designing an accurate classifier.

The use of computational intelligence (CI) methods (Bezdek, 1994; Bezdek, 1992; Pedrycz, 1997) for pattern recognition has often been shown to be very effective since it combines in synergy three very powerful computational tools:

- Neural networks, for representing highly non-linear relationships (Bishop, 1995; Rumelhart *et al.*, 1986).
- Fuzzy sets, for dealing effectively with imprecise or gradual boundaries between different classes of patterns (Pedrycz and Gomide, 1998; Zadeh 1997; Zadeh 1996; Zadeh 1979).
- Evolutionary computation, for solving global optimization of highly complex problems (Goldberg, 1989; Holland, 1975).

Neural networks are computational structures inspired by the human brain. A neural network is composed of a number of simple non-linear processing elements that are highly interconnected through a set of parameters (weights). These weights can be modified to achieve a desired functionality through a *learning* process. The basic properties that make neural networks attractive for pattern recognition tasks are their ability to learn from examples, to generalize and to represent highly non-linear relationships.

Fuzzy sets can be seen as an extension of ordinary sets theory. While in ordinary sets an element can either "belong" or "not belong" to a set, in fuzzy sets elements are characterized by their *grade of membership* in

the fuzzy sets. Algorithms related to fuzzy sets theory were shown to be successful when applied to solve problems of fuzzy uncertainty as it often occurs in pattern recognition tasks.

Evolutionary computation encompasses search methodologies (genetic algorithms, evolutionary strategies and genetic programming), which are biologically inspired by the process of natural evolution and are intended to solve global optimization problems.

Various combinations of CI methods have been applied in synergy to solve complex problems and were often demonstrated as more efficient than when using them individually. Pattern recognition of high dimensional data is one type of complex problems, and this thesis' focus is in that context.

This thesis has three objectives:

1. To study the potential of a genetic algorithm (GA) - based method for feature space generation. A GA is used to optimize piecewise approximation of data. Initially the method's validity (in terms of accuracy) is affirmed using synthetic and real-world data. Secondly this GA-based approach is applied to a pattern classification problem as an efficient feature extraction method, achieved through piecewise linear approximation. The method is evaluated thoroughly for classification of high-dimensional, biomedical data.
2. To present and investigate the concept of a fuzzy adaptive logic network (FALN). Being composed of two major computational units, perceptrons and fuzzy neurons, the FALN possesses unique properties of both transparency and computational aspects. The perceptrons represent a collection of hyper-planes and the combination of fuzzy neurons create clear logic combinations of the hyper-planes. The FALN's learning capabilities are shown and evaluated through numerous data sets.
3. A series of experiments using ensemble methods of FALNs is carried out in an attempt to demonstrate their effectiveness. Ensemble methods are generic methods that involve combining several classifiers to obtain a result of improved accuracy. These methods have been applied in numerous experiments, which were published in recent years, to several types of classifiers. This study examines the effect of applying ensemble methods to FALNs using two types of data sets.

The outline of this thesis is as follows.

The thesis begins with an overview, in Chapter 2, of the topic of pattern recognition in conjunction with CI. This chapter gives a general picture of the main tasks that make up a typical pattern classifier, namely,

feature extraction and classification. Next, a summary of the fundamental methods of CI is presented, which include neural networks, fuzzy sets and evolutionary computation. The chapter offers a concise description of each of these building blocks as well as of various methods by which they can be combined in collaboration for pattern recognition applications.

Chapter 3 presents the diversity of data sets used in this thesis for illustrating, clarifying and demonstrating the various methods that were investigated. In addition it describes the experimental framework and performance evaluation methodology.

The focus of Chapter 4 is a newly devised method of feature extraction, from high dimensional biomedical spectral data, using a GA. The GA serves as an optimization tool for constructing an accurate, piecewise linear (as a special case of a polynomial) approximation of highly dimensional data, as a means for obtaining significant feature reduction. The piecewise approximation performance is demonstrated and thoroughly evaluated using an application of high-dimensional biomedical spectral data, and compared with bottom-up segmentation. This detailed comparison is followed by a utilization of the constructed piecewise linear approximation to a pattern classification problem. The piecewise linear approximator is envisioned as a feature extractor that is shown to achieve significant feature reduction while being able to represent trends in the data and maintaining classification performance. The capabilities of this feature extractor are compared to the extensively used principal component analysis (PCA) in terms of attained feature reduction and classification performance.

Chapter 5 provides a comprehensive account of a pattern classifier based on a fuzzy adaptive logic network. The proposed pattern classification architecture takes advantage of an important synergy between the geometric representations of data and fuzzy logic-oriented operations on those representations. It is shown how this synergy is made operational and translates into a coherent architecture for the classifier. The crux of the proposed topology lies in a collection of simple linear classifiers (perceptrons) being combined into a logically coherent topology. Perceptrons possess simple geometrical interpretations while processing based on fuzzy operators (AND and OR logic units - fuzzy neurons) results in highly transparent and interpretable results. When combined, forming a fuzzy adaptive logic network (FALN), they give rise to the computing construct that retains the advantages, namely, processing and interpretability, of these two paradigms of information processing. The FALN's structure is initially presented, followed by the detailed mathematical expressions of the learning algorithm, used in the training phase. Next, the important issue of structure determination and initialization is discussed and several approaches are proposed. Finally, the application of FALN to several classification problems is demonstrated and studied and the performance of FALN is compared to other well-known classifiers.

In Chapter 6 the issue of building ensembles of FALNs is investigated. The chapter begins with a review of *ensemble methods*, which refers to the concept of creating a collection of classifiers in order to combine their individual results into one result, the motivation being that the combined result is often more accurate than that of the single classifier. Specifically, this chapter deals with *bagging* and *boosting*, being two commonly used methods for creating ensembles of classifiers. The principles and properties of bagging and boosting are reviewed along with references to previous studies done using these methods. It is worth noting at this point that the main experience accumulated with ensemble methods so far is from creating ensembles of either neural networks or of classification trees; however, described here is a first attempt to combine FALNs in an ensemble. In particular, this study is done with two applications:

- The first problem belongs to the area of quantitative software engineering, in which a data set of software metrics is viewed as a classification problem for the purpose of predicting the number of faults in a software module.
- The second study concerns classification of high dimensional biomedical data, and the results obtained with the ensemble methods are compared to those recorded earlier in the chapter that deals with FALN – Chapter 5.

The classification results of the ensembles, using the bagging and boosting methods are compared with those of a single FALN. A detailed analysis is provided in this chapter.

Figure 1 summarizes the thesis outline graphically. It illustrates the roadmap according to which the research of this thesis was carried out. This diagram shows the links between the various approaches that were investigated (CI related and generic), the diversity of data sets (synthetic, biomedical, software) and the achieved results.

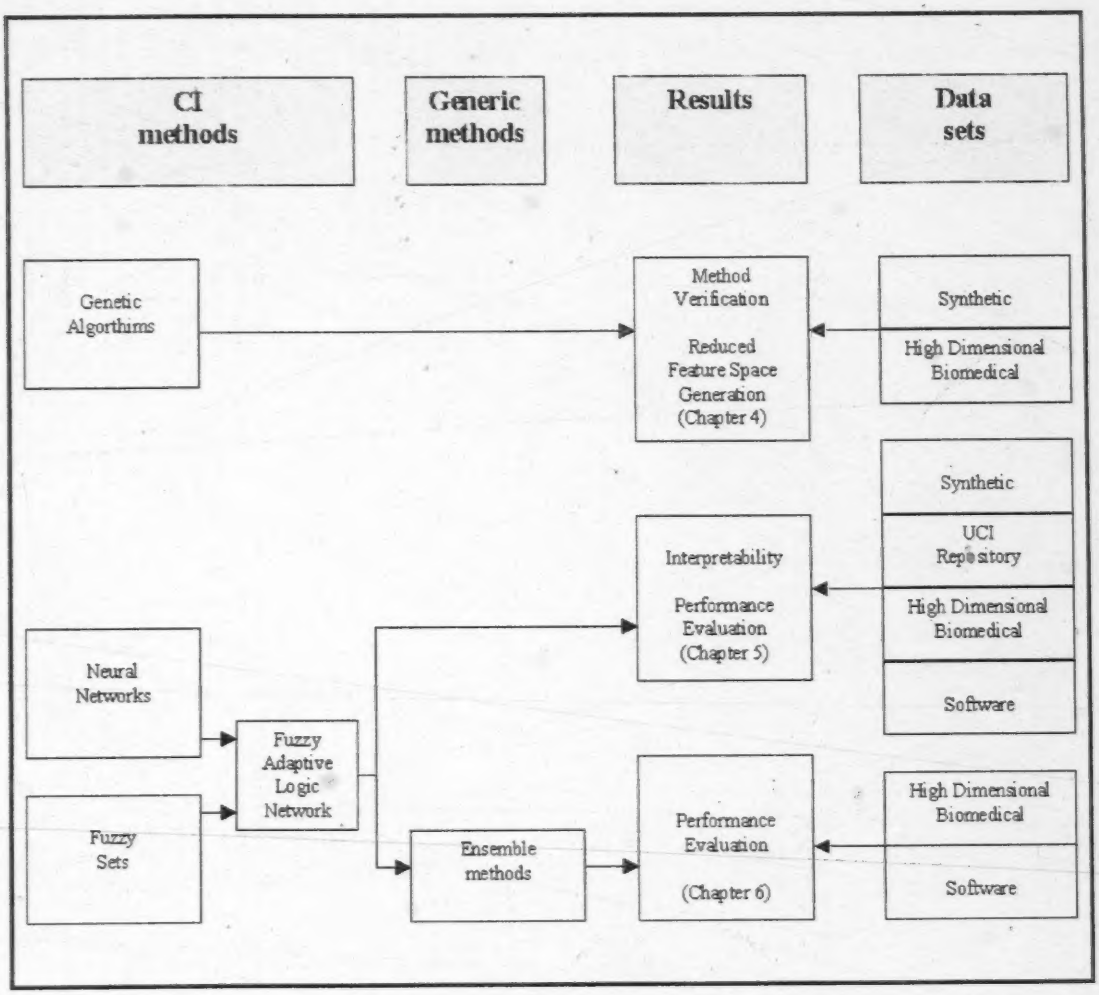


Figure 1: Thesis roadmap of empirical analysis

2. Computational intelligence and pattern recognition – an overview

2.1 Introduction

Computational intelligence (CI) (Bezdek, 1994; Bezdek, 1992; Pedrycz, 1997)is a growing field, which emerges out of synergy between several powerful information processing technologies:

- Neural networks
- Fuzzy sets theory
- Evolutionary computation (comprising genetic algorithms, evolutionary strategies and genetic programming)

Commonly CI (Bezdek, 1994; Bezdek, 1992; Pedrycz, 1997) is defined as a computing methodology that deals only with numerical data, has a pattern recognition component, exhibits an ability to learn and does not use explicit human knowledge. CI encompasses practical adaptation models, algorithms and implementations that pave a way for making intelligent decisions and predictions in complex environments.

2.2 Phases in pattern recognition

Typically a complete pattern recognition system is represented by several phases as illustrated in Figure 2 (Bishop, 1995; Duda *et al.*, 2001).

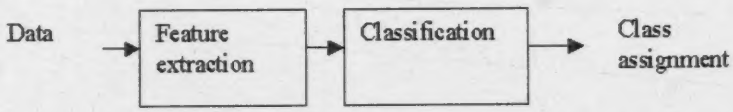


Figure 2: Pattern classification phases

Note that it is assumed that data is acquired previously and this data acquisition phase is not considered in this thesis. The collected data can be an image in a medical application, a segment of speech or various data associated with many other areas. The purpose of the first stage can be described as generating a set of features (using the raw data acquired before) that is the most appropriate for performing the classification function. Ideally this stage should lead to significant reduction in the amount of data, while extracting features that contribute the most for distinguishing between data classes. The second stage is the actual

process of assigning a class label to the pattern. The classification phase is the one that decides to which class the pattern belongs and labels it. This study deals with the two stages emphasized in Figure 2.

2.2.1 Feature extraction

The purpose of feature extraction is: (a) To transform the input features to some other space in which the classes will be more easily separable. (b) To achieve significant reduction of the number of features, making the classification more feasible.

Selection of a feature extraction method can be a crucial factor in achieving high recognition performance. Many methods of feature extraction and feature reduction have been developed, such as the Karhunen-Loeve transform (Duda *et al.*, 2001), also known as principal component analysis, Fourier transform, wavelet transform, piecewise polynomial representation, linear predictive coefficients, etc. Each method requires individual appropriate tuning and optimization according to the specific problem at hand.

2.2.2 Classification

Classification is the process using the feature vector provided by the feature extractor to assign the pattern to a category. A very basic classifier for assigning a feature vector to 1 of 2 classes is the linear classifier (Duda *et al.*, 2001) in which the output is computed by

$$y = \sum_{i=1}^{n} w_i x_i + w_0, \quad w_o, w_i, x_i \in \mathcal{R}, \; i=1..n \tag{1}$$

where **w** denotes a set of weights, $\mathbf{w} = [w_0\ w_1\ w_2 \ldots w_n]^T$, ($w_0$ is usually referred to as *bias*) and **x** represents the set of input features, $\mathbf{x} = [1\ x_1\ x_2 \ldots x_n]^T$, and the output y is a linear combination of the input features. The weights are determined during a training phase, using a training data set. Among the best-known methods of obtaining the weights are the perceptron learning rule, Widrow-Hoff (iterative methods) and the pseudo-inverse method (direct solution).

Of course, a plethora of other types of classifiers have been developed and researched for various problems, and in recent years many are involved with the use of CI methods. The following sections describe their fundamentals.

2.3 Computational intelligence building blocks

This section offers a brief description of each of the three CI processing technologies.

2.3.1 Neural networks

Neural networks (Bishop, 1995) have been widely used in recent years for pattern recognition. They are composed of highly distributed non-linear processing elements that operate in parallel and have the ability to represent non-linear and multi-variable relationships. Neural networks have the capability of learning from examples. Learning is achieved through a training process, during which the neural network (NN) is repeatedly presented with the set of input patterns and their associated desired outputs. The NN's parameters (weights) are updated (e.g. using gradient methods) until the required performance has been reached. After learning from a set of examples the NN has the ability to generalize to unfamiliar cases, and that makes it attractive for pattern recognition problems. Following is a brief description of two specific neural network implementations.

Feed-forward neural network

A linear classifier is often insufficient for achieving good enough separation between classes. One type of widely used classifier that can achieve representations of highly non-linear functions is the multi-layer feed-forward neural network (Bishop, 1995; Duda *et al.*, 2001), shown schematically in Figure 3. The network is composed of layers of neurons, where each of them performs a non-linear function (denoted by "f" in the figure) of the weighted-sum of its inputs. The number of layers and the number of neurons per layer is usually determined empirically. The weights are determined during the training phase, where various training algorithms are possible. Among those are: the error-back-propagation (Rumelhart *et al.*, 1986) (gradient method) and the Levenberg-Marquardt algorithm (Hagan and Menhaj, 1994) (performs non-linear least-squares optimization). Neural networks can be designed to solve not only the classification process, but also other input-output mappings. In one such application (Jin and Ma, 2000) a NN is used to perform efficient feature reduction

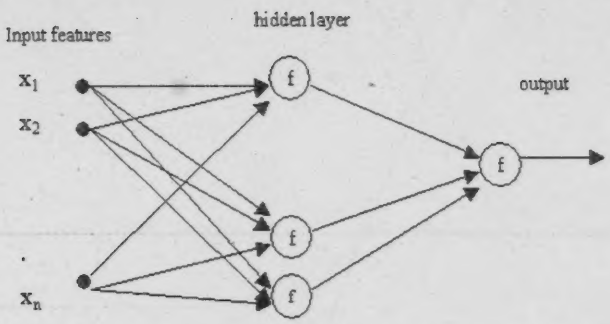


Figure 3: Feed-forward neural network classifier with one hidden layer

Adaptive logic network (ALN)

ALN (Armstrong *et al.*, 1995; Cogger, 1997) is a relatively new type of NN. The basic computational element of an ALN is the perceptron governed by an expression similar to (1), i.e. a linear function. An ALN incorporates a collection of linear functions by a tree of nodes that perform *minimum* and *maximum* operations. This unique structure actually creates a piecewise linear approximation of the given data and enables representing any continuous function at any desirable accuracy (Armstrong *et al.*, 1995). One of the ALNs distinct advantages over past approaches of NNs is that it is far more computationally efficient. The *adaptive* term in an ALN refers to the adjustable parameters in an arbitrary number of linear functions used.

2.3.2 Fuzzy sets

Fuzzy sets are a generalization of conventional set theory (Zadeh, 1997; Zadeh 1996; Zadeh 1979) and were introduced as a mathematical means of expressing vagueness or fuzziness that we are very used to dealing with in everyday life. In conventional set theory it is very clear whether a certain element belongs or does not belong to a specific set, whereas in fuzzy sets a term called *degree-of-membership* plays a key role. This membership value expresses the degree to which an element is compatible with a set. The membership value can be between 0 and 1 (while for conventional sets the value is of boolean nature - either 0 or 1).

Fuzzy sets (FS) theory in the context of pattern recognition (Chi *et al.*, 1996) is based on the notion that boundaries between different classes and patterns can be imprecise, or gradual (fuzzy). An important characteristic of fuzzy models is that they are based on partitioning information into fuzzy regions by means of fuzzy sets. The features that make up the pattern undergo a degree-of-membership determination stage. Such membership functions could be for example - *low*, *medium* and *high*. This way of feature representation is the basis of knowledge representation in fuzzy models. The features are translated into a

more linguistic form and that enables knowledge to be characterized in a more examinable manner with the use of fuzzy linguistic "if-then" rules.

Evidently when considering practical aspects one realizes that there are vast possibilities of designing membership functions. There is the question of determining how many membership functions to create for each of the input and output variables, along with their shapes and parameters. Figure 4 is a simple illustration of a possible representation of a variable that can have values in [0,1], using four triangular membership functions denoted by: A1, A2, A3 and A4. Say that these membership functions represent "linguistic values" of: *very low*, *low*, *high* and *very high*, respectively. Then a value of 0.1 would have a membership value of about 0.7 to *very low* and a membership value of about 0.3 to *low* (the membership values to *high* and *very high* would be 0). That is one example where computational aspects come into effect and each application has to be studied and dealt with individually. This topic is strongly related to *rule-based computations* (Pedrycz and Gomide, 1998).

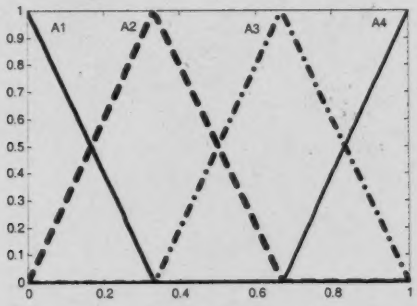


Figure 4: Example of partitioning to four triangular membership functions

Another computational model is *fuzzy neurocomputation* (Pedrycz and Gomide, 1998), which hinges on the concepts of logic operations of fuzzy sets. At the basis of this computational model lie the definitions of *fuzzy neurons*, OR and AND neurons. A more detailed definition and characterization of fuzzy neurons is provided in the sequel.

2.3.3 Evolutionary computation

Evolutionary computation (EC) methods are inspired by the process of natural evolution (Chambers, 2000). Different evolutionary processes have been proposed: GAs, genetic programming and evolution strategies. The most popular models are GAs (Beasley *et al.*, 1993; Chambers, 2000; Goldberg, 1989; Holland, 1975;

Michalewicz, 1992). A GA is a method for solving optimization problems through a search procedure. By utilizing the theories of natural evolution and natural selection, GAs are able to evolve solutions for complex problems if the problem is encoded suitably into the basic element of the GA – the *chromosome*. A chromosome is made of *genes*. A widely used method is the binary representation in which the chromosome is represented by a string of binary digits.

A GA has four main elements:

1. The genetic code, a concise representation of *phenotype* (collection of parameters) by *genotype*.
2. The *population*, a number of individual solutions.
3. The *fitness* function, an evaluation of the usefulness of an individual.
4. The creation of a new population, a set of methods for producing the next generation:
    - Selection: a set of individual chromosomes is selected where the probability of being selected is based on their fitness.
    - Crossover: two new chromosomes are created using two randomly selected existing chromosomes.
    - Mutation: Create new chromosomes from randomly selected existing chromosomes.
    - Elitist strategy: The best chromosome (of the highest fitness value) is always passed on to the new population.

The GA works as follows. First, a population of individuals is generated randomly. The fitness of each individual is then evaluated, and highly fit individuals are selected to generate a new population – the next generation. The cycle of evaluation, selection and new generation creation, continues until a satisfactory solution is found. This process is illustrated in Figure 5. It should be emphasized that appropriate representation of the parameters by a genetic code and the choice of a suitable fitness function can be a crucial factor in reaching efficient solutions.

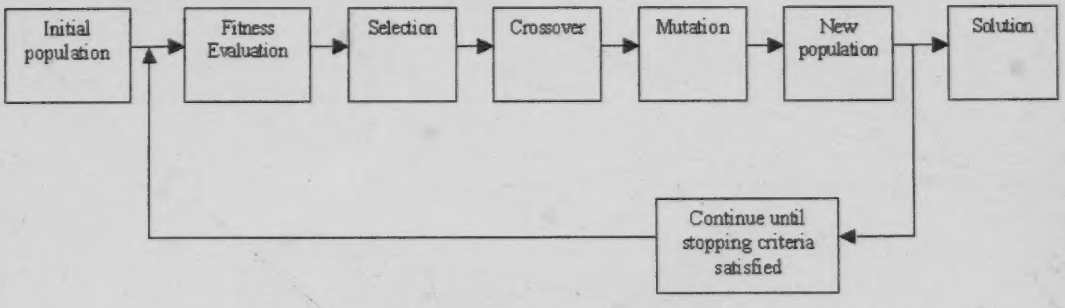


Figure 5: Main concept of genetic algorithm

2.4 Computational intelligence for pattern recognition

Schematically, CI can be depicted by Figure 6 below.

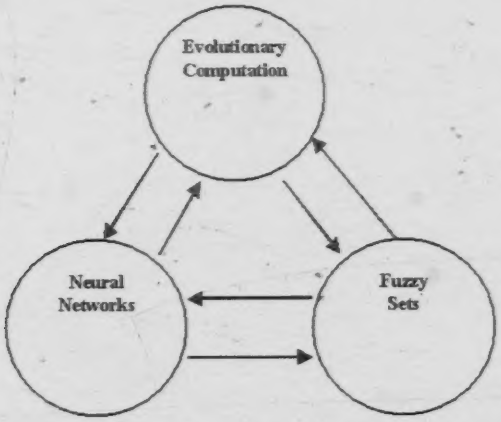


Figure 6: Computational intelligence: synergy of neural networks, fuzzy sets and evolutionary computation

Introduced in this section are several basic concepts of collaboration between the CI building blocks in conjunction with pattern recognition.

2.4.1 Neural networks and fuzzy sets

Rule generation from artificial neural networks (Mitra and Hayashi, 2000), is gaining in popularity due to its capability of providing some insight to the user about the symbolic knowledge embedded within the network. Fuzzy sets are an aid in providing this information in a more human comprehensible or natural form, and can handle uncertainties at various levels. Both FS and NN are computational approaches to modeling expert behavior. The goal is to mimic the actions of an expert who solves complex problems. In other words, instead of investigating the problem in detail, one observes how an expert successfully tackles the problem and obtains knowledge by learning. If one has knowledge expressed as linguistic "if-then" rules, one can build a fuzzy system. On the other hand, if one has data or can learn from a simulation of the real task, a NN is more appropriate. The advantages of both neural networks and fuzzy systems can be integrated in a neuro–fuzzy approach. The two important ways of combining neural and fuzzy computation are described in the following.

Adaptive neuro-fuzzy inference system

Fuzzy neural network (FNN) (Jang *et al.*, 1997; Mitra and Hayashi, 2000) is a neural network equipped with the capability of handling fuzzy information. In an ordinary NN the training process involves adaptation of the network weights. In an FNN, input features are translated into membership values of fuzzy sets (typically linguistic values such as low, medium, and high). The initialization of the membership function is the first and essential step and can be done, for example, using statistical methods or clustering methods. During the training phase of an FNN, the strong optimization capability offered by NNs is used for also tuning the membership function parameters of the fuzzy rule-based system. This is done usually using gradient-based methods and requires that the membership function would be differentiable. Similarly, methods of translating the connection weights and the outputs of the NN to membership values have been proposed.

Incorporating fuzzy logic into neurons

Neural networks with fuzzy neurons are capable of processing fuzzy information. Logic based NNs (Pedrycz and Gomide, 1998) incorporate the idea of having neurons that have responses based on FS theory. The responses are similar to those of NOT, AND, and OR functions in Boolean logic. The learning process is gradient-based.

2.4.2 Evolutionary computation and neural networks

Combinations of GAs and NNs (Schaffer *et al.*, 1992) can be basically divided into two types: supportive (used sequentially) and collaborative (used simultaneously).

Supportive combinations

Supportive combinations consist of using one of the methods to prepare data for use of the other, such as using a GA to select features for use by the NN classifier (the most common use). Another scheme mentioned in the literature is using a GA to determine the learning parameters of a NN (learning rate and momentum).

Collaborative combinations

In collaborative combinations, most commonly the GA can be applied to determine the NN topology, or weights or both. Determining the weights involves optimizing the weights of the NN for a given topology. Each member of the population specifies a complete set of weights for the NN. Fitness is then calculated by summing the squared errors over a set of training data. The alleged advantages of this approach is that the ability of GA to perform a global search helps in avoiding local minima, and that a GA can be applied to

problems in which the error gradient is difficult or costly to obtain.

In genetic programming (GP) for logic-based NNs (Gaudet, 1996) the idea is similar to that of a GA, but instead of operating on a string of binary digits, the algorithm operates on a tree which represents the parse tree for a given program.

2.4.3 Evolutionary computation and fuzzy sets

The most widely used combination in this context is to use a GA to optimize the rule-base for the classification process (Ishibuchi *et al.*, 1996). A major problem that arises when using fuzzy rule-based systems with high dimensional data is that the number of rules can be very large. For example, in case of six linguistic values, for each axis of a 13-dimensional input-space, the total number of possible if-then rules is 6^{13}, which is 13 billion. The need for reducing the rule-base drastically is obvious.

In order to optimize a fuzzy classification system with linguistic if-then rules, first a large number of linguistic rules are generated from the training data, and then only a small number of significant rules are selected by a GA in order to construct a compact, fuzzy classification system. A set of linguistic rules is coded as an individual in the GA. The number of correctly classified training patterns and the number of selected linguistic rules determines the fitness.

2.5 Summary

This chapter discussed and reviewed the basic approaches investigated for applying CI methods to pattern recognition. The basic phases in pattern recognition were described followed by a separate brief portrayal of each of the computational tools included in CI. Concluding this chapter is a concise account of several of the most commonly researched methods of combining CI methods in collaboration for solving pattern recognition problems.

3. Data and methodology

3.1 Introduction

This chapter (a) discusses the details of the diversity of data sets analyzed in this thesis, (b) outlines the evaluation methodology of the performance achieved with the different methods that were developed.

3.2 Data sets

A variety of data sets are used throughout this study. Several synthetic 2-dimensional data sets are used for illustration purposes. The growing interest and need in providing accurate diagnoses based on biomedical data, often highly dimensional, motivates the investigation of a few such data sets here. In quantitative software engineering data, the ability to provide interpretable prediction of software reliability is of major importance and is also dealt with. Additionally, two examples taken from the UCI Machine Learning repository (Murphy and Aha, 1994) are used for analysis and comparison.

3.2.1 Synthetic data

Spiral data

The two-dimensional spiral patterns that are quite frequently treated as a test-bed for neural networks is governed by the description given in polar coordinates

$$x = \gamma\cos\theta;\ y = \gamma\sin\theta \quad \text{(spiral \# 1)}$$
$$x = -\gamma\cos\theta;\ y = -\gamma\sin\theta \quad \text{(spiral \#2)} \qquad (2)$$

where $\gamma = \frac{1}{\pi}(\theta + \frac{\pi}{2})$ and $\theta = \frac{k\pi}{16}$ where k =1, 2, …, N with "N" being the number of patterns in each class. These patterns are visualized in Figure 7.

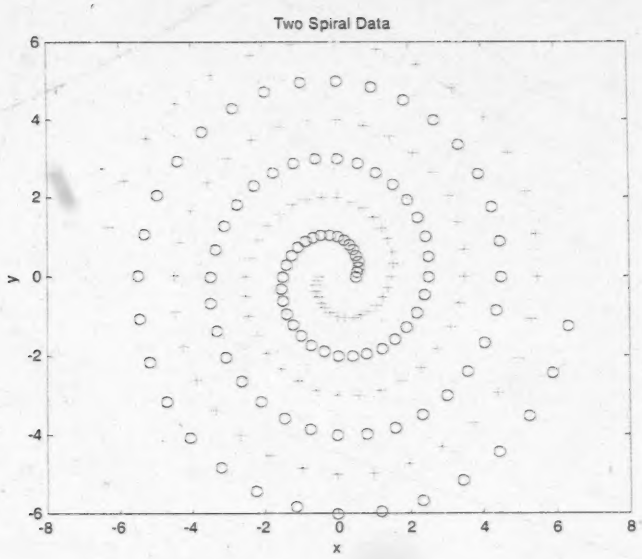
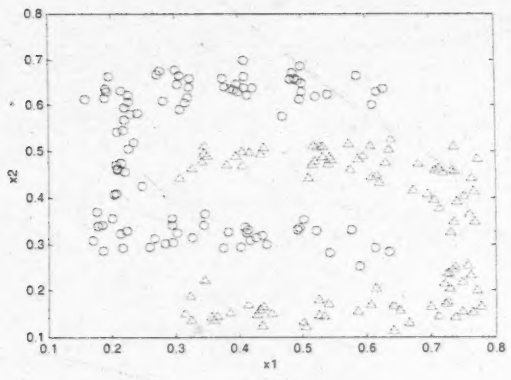


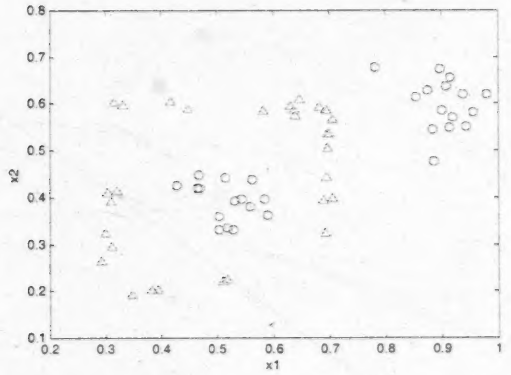
Figure 7: A two-class spiral data set (N=96)

Non-linear - 2-dimensional data sets

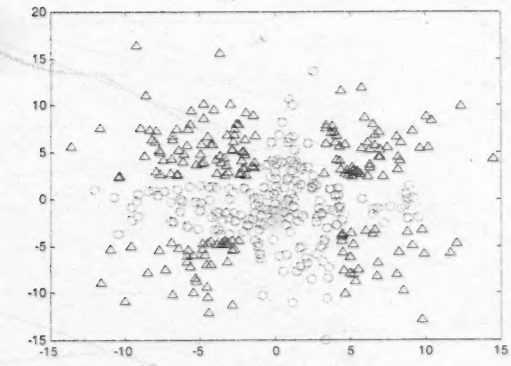
Several data sets illustrated in Figure 8, are presented and dealt with in the sequel. They are all 2-class and 2-dimensional and apparently a non-linear decision boundary is required to achieve perfect separation between the classes.



(a)



(b)



(c)

Figure 8: Non-linear data sets: piecewise linearly separable data (a), clustered data (b), highly structured data (c)

3.2.2 High dimensional biomedical data sets

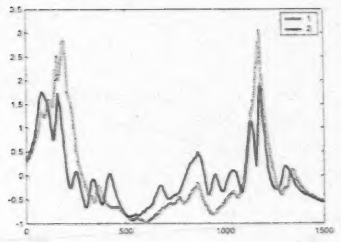
This section describes biomedical data sets associated with various diseases. It is important to note that successful classification of such datasets could lead to early detection of the associated disease states. In some cases, the disease state can be detected (using the MR and IR spectra) at a biochemical level before morphological changes occur at the cellular level. This is very important for diseases such as cancer, since treatment is more effective the earlier the disease is detected. These data sets are especially interesting and challenging due to their high dimensionality.

Candida species

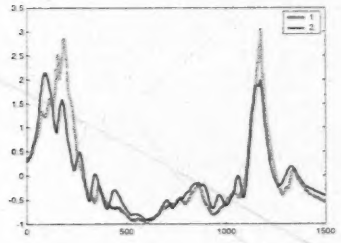
MR spectra of isolates of five different Candida species were acquired: Candida albicans (AL), Candida parapsilosis (PA), Candida krusei (KR), Candida tropicalis (TR), and Candida glabrata (GL). The yeast colonies were suspended in saline. The magnetic resonance spectra were acquired at 37°C on a 360 MHz MR spectrometer. Each of the 444 spectra contained 1500 features and were divided into five classes, according to the corresponding Candida species: 104 AL, 91 GL, 81 KR, 93 PA, and 75 TR. The objective defined with this data set was to classify the AL data vs. each of the other four Candida species. Using this data, four 2-class classification problems were set up, where for each of the data sets, a training set and a testing set are labeled as outlined in the following.

- AL vs. GL (60 spectra from class 1, and 60 from class 2 are used for training; 44 / 31 spectra used for testing)
- AL vs. KR (54 / 54 spectra used for training; 50 / 27 spectra used for testing)
- AL vs. PA (62 / 62 spectra used for training; 42 / 31 spectra used for testing)
- AL vs. TR (50 / 50 spectra used for training; 54 / 25 spectra used for testing)

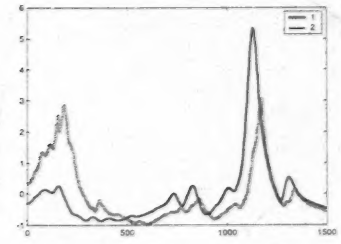
Following, Figure 9 depicts plots of representative examples of the two classes from each of the classification problems.



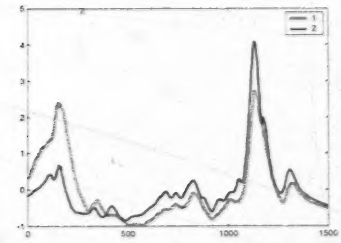
(a)



(b)



(c)



(d)

Figure 9: Plots of typical samples of two classes from each of the biomedical data problems; AL vs. GL (a), AL vs. KR (b), AL vs. PA (c), AL vs. TR (d)

Infrared (IR) spectra of synovial joint fluid

The spectra are divided into one of three classes: 72 samples of rheumatoid arthritis, 72 samples of osteo-

arthritis, and 42 control samples. Each The IR spectra cover the range 1000–3700 cm^{-1} and were discretized to 2801 data points. Some of the typical IR spectra are shown in Figure 10.

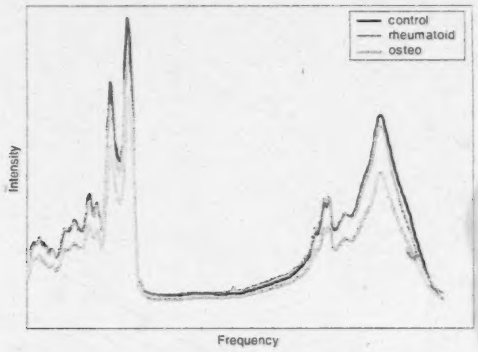


Figure 10: Examples of biomedical data: IR spectra of synovial fluid

Magnetic resonance (MR) spectra of tissue biopsies from brain neoplasms

The spectra fall into one of three groups: 95 spectra that correspond to meningiomas, 74 astrocytomas samples, and 37 spectra corresponding to non-tumourous brain tissue that served as control spectra. The spectra were acquired in the range 0.3–4.0 ppm, discretized to 550 data points and area-normalized. Typical MR spectra from that data set are shown in Figure 11.

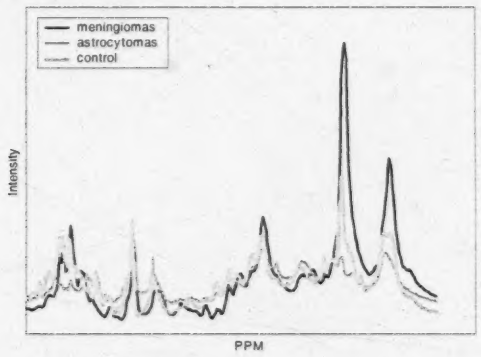


Figure 11: MR spectra of brain neoplasms

MR spectra of a biological fluid

This data set is discretized to 512 points. The spectra were divided into one of three classes: 108 samples belonging to the class of normal patterns, 54 being of borderline character, and 65 abnormal patterns. Examples from the three classes are displayed in Figure 12.

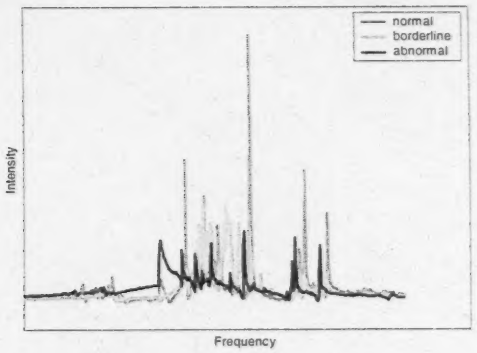


Figure 12: MR spectra of a biological fluid

3.2.3 Software engineering data

It is instructive to elaborate here on the software data being used in this case study; the Medical Imaging System (MIS), (Lind and Vairavan, 1989; Munson and Khoshgoftaar, 1996) is a commercial software system consisting of approximately 4500 routines written in about 400,000 lines of Pascal, FORTRAN and PL/M assembly code. The MIS development took five years, and the system has been in commercial use at several hundred sites for quite a few years. The MIS data were collected as the number of changes made to each module due to faults discovered during system testing and maintenance over an observation period of three years. Along with the above parameter, 11 software complexity measures (metrics) were recorded; refer to Table 1. In this study, MIS is represented by a subset of the whole MIS data with 390 modules written in Pascal and FORTRAN. These modules consist of approximately 40,000 lines of code. The goal is to develop a prediction model of software quality in which the number of modifications (changes) is projected on a basis of the values of the 11 software metrics that are used to characterize a software module.

Measure	Description
Changes	Number of changes
LOC	Number of lines of code including comments, declarations and the main body of the code
CL	Number of lines of code, excluding comments
TChar	Number of characters
TComm	Number of comments
MChar	Number of comment characters
DChar	Number of code characters
N	Halstead's Program Length $N = N_1+N_2$, N_1 is the total number of operators, N_2 is the total number of operands
N'	Halstead's Estimate of Program Length $N' = n_1 \log_2 n_2 + n_2 \log_2 n_2$, n_1 is the number of unique operators, n_2 is the number of unique operands
NF	Jensen's Estimate of Program Length Metric $\log_2 n_1! + \log_2 n_2!$
V(G)	McCabe's Cyclomatic Complexity Metric, where V(G)=e-n+2, and e represents the number of edges in a control graph of *n* nodes
BW	Belady's bandwidth measure $BW=1/n \Sigma_i i L_i$, L_i represents the number of nodes at level i in a nested control flow graph of n nodes. This measure indicates the average level of nesting or width of the control flow graph representation of the program

Table 1: Description of the MIS data set with a detailed characterization of the software measures (adopted from Munson and Khoshgoftaar, 1996)

3.2.3 Machine learning repository data

Two data sets from the UCI Machine Learning repository (Murphy and Aha, 1994) of different levels of difficulty are used in this thesis. These well-known data sets are commonly used in studies for evaluating classifiers.

Wine data set

This data set consists of the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines. In total there are 178 samples in this data set, 59 samples are of class no. 1, 71 belong to class 2 and 48 samples – to class 3.

Pima Indian diabetes data set

This data set includes several characteristics taken from a population of female patients, at least 21 years old, of Pima Indian heritage near Phœnix, Arizona. The diagnostic investigated is whether the patient shows signs of diabetes according to World Health Organization criteria. This 2-class data set originally consists of 768 samples of 8 features and Ripley (1996) reduced it to 532 samples of 7 features, omitting

instances with missing attributes.

3.3 Validation methodology

In many classifier design problems, for evaluating the performance of a classifier, the designer of the experimental framework defines a set of data for producing the classifier and a separate set of data for testing it. That is the approach taken in the case of the Candida species classification problems. The data sets were split in a deterministic manner into a training set and a testing set. This approach is not uncommon and may originate from the specific requirements and goals of the designer involved with the specific data at hand.

Classifier evaluation with other data sets used in this study was performed using a different approach, since these data sets were not initially split into a training set and a testing set. To gain a consistent evaluation of the performance of the FALN and the other classifiers to which the FALN is compared, these experiments were carried out with the 5-fold cross validation technique. 5-fold cross validation was employed as follows. The data is split randomly, incorporating stratification, into two sets: (a) a training set, which consists of 80% of the data examples (b) a testing set consisting of the remaining 20%. The classifier is then trained using the training set and when training is completed its performance is evaluated using the testing set. For training algorithms that are based on *early stopping*, the training set is split randomly again into 60% (on which actual training is performed) and 20% (validation set). The purpose of this is to avoid over-fitting and it is done by constantly testing the classifier on the validation set, during the training process. When it appears that the classifier's generalization ability (estimated through its performance on the validation set) starts to deteriorate, training is halted. The assumption made here is that the classifier's performance on the validation set is indicative of its generalization capability and therefore training is stopped accordingly. This mechanism is known as early stopping. This training-testing procedure is repeated five times (hence it is termed 5-fold) while ensuring that the five testing sets, if unified, include all examples in the data set. In each of the five experiments the classifier's performance is recorded with regard to the training and testing set and the final results are given in terms of the respective mean values and standard deviations (SD) of these five experiments.

3.4 Summary

This chapter described in detail the various data sets dealt with in this study and their objectives. The chapter also provided the details of the performance evaluation methods used, specifically the 5-fold cross validation scheme.

4. GA-based feature space generation

4.1 Introduction

Functional piecewise approximation seeks data representation that is compact, highly simplified and meaningful. This chapter presents a GA-based approach for computing a piecewise polynomial representation of data, with the focus being on piecewise linear approximation in an application of biomedical spectral data. The application of this method as a feature extraction method for classification of a set of feature vectors, specifically biomedical spectral data, is also investigated.

4.2 Biomedical data – Candida species

Biomedical spectra obtained by MR spectroscopy are typically of high dimensionality and few available samples. A statistically meaningful analysis of a limited number of high-dimensional data points, specifically for classification purposes, involves great difficulties, on account of the tremendous sparseness of high-dimensional spaces. This issue if often termed the *curse of dimensionality* (Bellman, 1961) which relates to many undesired outcomes such as one that often arises, that a large number of features can lead to a reduction in the performance of a classifier. This is largely due to the fact that the number of training data required in order to specify a mapping grows exponentially with the number of features (Bishop, 1995; Duda *et al.*, 2001). This intricacy is additional to the apparent severe requirements that a high-dimensional problem might create concerning large memory in conjunction with long computation time, and possible numerical issues. In cases where many of the features are irrelevant, the resulting classifier may behave comparatively deficiently since the input dimension is high, and it uses many of its resources to characterize irrelevant portions of the input space. All these difficulties associated with highly dimensional data make feature extraction a necessity in cases of this type.

One such typical data set is the Candida species data set presented earlier in Chapter 3. While each feature vector is 1500 points long, there are only 444 samples available.

4.3 Piecewise curve approximation as a model of a feature space: analysis of approximation capabilities

The area of piecewise linear approximation has been researched in the past four decades approximately, and the method presented here is compared with another well-known method, bottom-up segmentation. The expansion of this method to piecewise polynomial representation is shown to be straightforward.

4.3.1 Background

A curve piecewise approximation algorithm receives a series of "n" points $\mathbf{y} = [y_1, y_2, \ldots, y_n]$, $y_i \in \mathcal{R}$ and produces an approximation by breaking up the series of point into "m" segments, where in each segment the data is approximated by a function of the x-axis points. In the cases analyzed here, we assume that the x-axis points are actually indices to **y**, i.e.: [1,2, …, n]. Evidently, it is desirable that m would be as small as possible (and much smaller than n), and the function representing the data per segment could be expressed compactly using a small set of parameters. In the case of piecewise linear approximation, each segment of points is represented by a line of the form $\bar{y_i} = ax_i + b$, (x_i define the x-axis segment points), which can be expressed using two parameters (a and b) thus resulting in a representation consisting of 2m parameters instead of n points. Additionally, the goal is to have an approximation that is accurate to a certain measure. Many solutions have been proposed to a large variety of problems. The topic of curve approximation (also referred to as curve fitting or curve simplification), specifically using piecewise methods (linear or more generally – polynomial) has been addressed extensively since the 1960's (Saupe, 1997). The subject is encountered in various research areas, such as:

- Mathematics (numerical analysis, optimization).
- Engineering (pattern recognition, signal processing, data compression).
- Computer science (computer vision, computer graphics, cartography).

Curve piecewise approximation aims at various applications, i.e. biodiagnostics, robotics and many more. A comprehensive survey of these methods is beyond the scope of this thesis, however, properties associated with the various methods will be mentioned here:

- The approximation quality evaluation criterion.
- The type of constraints imposed on the approximation – whether the split points should be restricted to a grid or not, does the approximation have to be continuous or not, whether the values at the break points should or should not coincide with the original curve.
- Some methods are suitable for fitting general curves in the two-dimensional plane (such as for shape recognition), while others are appropriate only for curves that represent functions or one-

dimensional data (time domain, spectra, etc.).

An example is the application of ECG signal compression, thoroughly described in (Nygaard *et al.*,1999), discussed also in (Keogh *et al.*, 2001), which examines segmentation of other time series data sets as well. Among the plethora of methods there is the FAN algorithm (Dipersio and Barr, 1985), where the end point of the line segments are restricted to be from the original data, however), it is not considered optimal in the sense of producing a minimal number of segments in order to achieve the uniform error requirement. Imai and Iri (1986) present an optimal method for approximating a piecewise linear function by another piecewise linear function such that the magnitude of the approximation error is bounded by a pre-defined value. Another extensively used method is the Douglas-Peuker (DP) algorithm (Douglas and Peuker, 1973; Duda and Hart, 1973; Heckbert and Garlan, 1997; Hershberger and Snoeyink, 1992; White, 1985), which attempts to represent a curve using a series of lines in accordance with a pre-defined tolerance constraint.

Segmentation methods

Keogh *et al.* (2001) discuss piecewise linear representation in the context of data mining time series databases. Three major approaches to segmentation are reviewed, namely:

- Sliding Windows: The Sliding Window principle of operation is to anchor the first point of a candidate segment at the first data point of the curve, and then attempting to approximate the rest of the data by increasing the segment until the approximation error exceeds the pre-specified allowed tolerance. Once that occurs, the segment is defined and the anchor is moved to the next point beyond the newly formed segment. The FAN algorithm mentioned earlier is a variation of this method.
- Top-down: The data is split in a recursive fashion until the pre-defined error criterion is achieved.
- Bottom-up: This method can be seen as the dual of the top-down method. The algorithm starts from the best existing approximation and then the segments are iteratively merged until the desired approximation accuracy is obtained. This is accomplished by calculating the cost of merging each pair of adjacent segments and iteratively merging the lowest cost pair at each stage. As concluded (Keogh *et al.*, 2001), the bottom-up method is mostly the more accurate comparing to either the sliding windows or top-down methods. This method is also provided in Matlab implementation (Keogh, 2003), which was used for evaluation purposes as described in the subsequent sections.

The three segmentation methods can be designed to produce either a continuous approximation (linear interpolation approach – connecting the end points of the segment) or a non-continuous one (linear

regression based – best in the least squares sense). The Matlab implementation (Keogh, 2003) used produces a non-continuous approximation as can be seen in Figure 13 showing the resulting piecewise linear approximation calculated with 2-10 linear segments using the bottom-up segmentation method. The upper-left plot shows the approximation with two lines etc. and the lower-right graph illustrates the result with 10 linear segments.

The pseudo-code for the bottom-up segmentation algorithm (first approach) is given in Table 2 below.

Create initial fine approximation

For all neighboring pairs calculate the cost of merging

Repeat

1. Find the lowest-cost pair to merge.
2. Perform merging
3. Update records
4. For all neighboring pairs calculate the cost of merging

Until approximation criteria is met

Table 2: Pseudo-code for the bottom-up segmentation algorithm

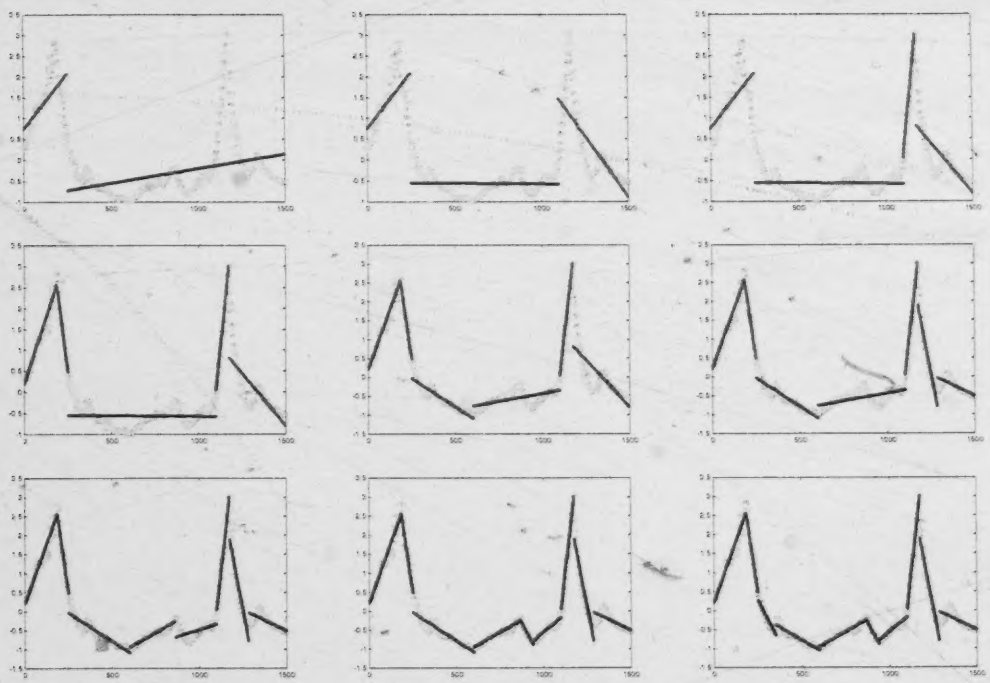


Figure 13: Nine approximations using bottom-up segmentation with different number of segments; illustrated on AL data spectrum example

GA-based algorithm

The alternative method that is investigated here is based on GA optimization (Beasley *et al.*, 1993; Chambers, 2000; Goldberg, 1989; Holland, 1975; Michalewicz, 1992). In particular, a real-coded GA (Herrera *et al.*, 1998) is used for optimizing the piecewise linear representation of the discussed biomedical data. A real-coded GA is based on real number representation as opposed to coding into a string of binary digits. In many applications that involve optimization in continuous search spaces it would seem the natural approach.

Genetic algorithms

A GA may be defined as a method for solving optimization problems through a non-deterministic search procedure that utilizes the theories of evolution and natural selection. By following the natural evolution principle, GAs are able to evolve solutions for complex problems. A GA has four main elements: the genetic code, a concise representation of phenotype by genotype; the population, a number of individual solutions; the fitness function, an evaluation of the usefulness of an individual; and the generation of a new

population, a set of methods for generating new individuals. The GA works as described:

First, a population of individuals is generated randomly. Next, the following steps are repeated iteratively as required.

- Selection: the fitness of each individual is then evaluated, and highly fit individuals are selected to generate a new population in the next steps. Several methods have been proposed and the one used here is tournament selection with size 3.
- Crossover: in this step the selected chromosomes are combined ("crossed over") to produce a new generation. The one incorporated here is one of the commonly used methods, namely arithmetic crossover.
- Mutation: this operation consists of randomly picking chromosomes (according to a pre-defined probability) and modifying them in a random fashion. In these experiments random mutation was used.
- Finally, elitist strategy is employed by placing randomly in the population, the best chromosome (with the highest fitness value) thus, producing the next generation in the GA operation.

The cycle of fitness evaluation, selection, mutation and new generation creation continues until a satisfactory solution is found. Choosing the appropriate way for representing the parameters by a genetic code and the suitable fitness function can lead to efficient solutions.

The issue of optimizing piecewise line fitting using GA was discussed previously (Pittman and Murthy, 2000; Pittman and Murthy, 1997), where the formulation of the piecewise approximation parameters into the GA's chromosome is done differently than explored here, leading to a continuous piecewise linear representation. Furthermore, in these previous studies (Pittman and Murthy, 2000; Pittman and Murthy, 1997) the GA is used as a method for achieving optimal breaking points in a single curve, while in this study a straightforward expansion to a set of data vectors is presented, allowing to achieve a set of breaking points common to the whole data set. This expansion is extremely useful for classification problems of highly dimensional data as an efficient feature reduction method that is capable of preserving and representing local trends in the data. A real-coded GA is explored here as a method for optimizing the breaking points of the curve for piecewise linear representation. Note that the main difference between the GA-based method and the segmentation methods is that the GA optimizes in parallel a collection of segmentation points while segmentation methods do it in sequence. It should be noted that restrictions such as: the approximation to be continuous, or that its values at the breaking points to be equal to the original data, do not apply to the approximations produced by neither the GA-based method, nor the bottom-up

segmentation method used in this study. Additionally, the GA-based method is expanded easily to perform piecewise polynomial (rather than linear) curve fitting, as demonstrated here.

The details of the GA are as follows.

Chromosome structure: in the context of the discussed approach, two basic chromosome structures may be considered.

- "Sorted indices": according to one option, a chromosome consists of locations at which the data vector is to be broken into sections. Additionally, these values are sorted and situated in the chromosome in ascending order. This simple operation was found to increase the GA's efficiency significantly; without it the crossover operation end up combining randomly high values from one chromosome with low values from another chromosome, consequently slowing the GA's convergence. For example, if the pre-determined number of segments were N the chromosome length would be N-1, which is the number of breaking points that define the edges of the segments. These nine values would all be in the range (1,L), where "L" is the data vector length.
- "Increments": the second approach (Gacek and Pedrycz, 2003) includes construction of a chromosome so that it includes elements where each stands for the distance between two breaking points, normalized to the entire interval of the curve. Thus all the chromosome elements are in (0,1), the first element represents the distance between the first point of the curve to the first breaking point, the second element represents the distance between the first breaking point to the second breaking point, and so on.

Fitness function: The fitness is calculated as follows. Since the chromosome defines breaking points of the original data into sections, these values are first rounded to produce vector indices and an additional step is taken in order to avoid having two identical indices, by incrementing one of the values by 1. Then in sequence, each pair of adjacent point is used to select the appropriate sector of the data, for which the best fitting linear segment is computed in the least squares sense. This operation is repeated for all neighboring breaking while accumulating the sum-square error for each segment in order to obtain the overall sum-square approximation error.

The basis of the segmentation process is to find the best fitting polynomial to given data points in the least-squares sense (Press *et al.*, 1992).

Tuning of the GA parameters

A series of initial experiments was carried out in order to determine a solid foundation for the GA-based optimization method, which consists of the selection of the chromosome structure as well as other

parameters namely, population size, number of generations, crossover rate and mutation rate. This basis served the extensive experiments conducted and described in the sequel. Tab. 3 and 4 summarize the results that were obtained using two types of synthetic data, and a sample taken from the biomedical spectra (AL example no. 1) used later on in this study. The artificial data used here were: a piecewise linear function made of eight segments, and a smooth function created by summing two sine functions.

Data	No. of seg.	Pop. size	No. of gen.	Xover prob.	Mut. Prob.	Error magnitude
PWL (8)	8	20	20	0.8	0.1	0.0244±0.0350
	8	20	50	0.8	0.1	0.0199±0.0358
	8	50	50	0.8	0.1	0.0137±0.0224
	8	100	50	0.8	0.1	0.0025±0.0084
	8	100	50	0.5	0.1	0.0159±0.0250
	8	100	50	0.9	0.1	0.0081±0.0202
	8	100	50	0.8	0.2	0.0133±0.0321
Smooth	10	20	20	0.8	0.1	0.0837±0.0723
	10	20	50	0.8	0.1	0.0715±0.0598
	10	50	50	0.8	0.1	0.0746±0.0569
	10	100	50	0.8	0.1	0.0593±0.0434
	10	100	50	0.5	0.1	0.0662±0.0509
	10	100	50	0.9	0.1	0.0610±0.0420
	10	100	50	0.8	0.2	0.0609±0.0445
AL no. 1	10	20	20	0.8	0.1	0.1131±0.1369
	10	20	50	0.8	0.1	0.1326±0.1404
	10	50	50	0.8	0.1	0.0997±0.0938
	10	100	50	0.8	0.1	0.0776±0.0794
	10	100	50	0.5	0.1	0.0853±0.0814
	10	100	50	0.9	0.1	0.0795±0.0847
	10	100	50	0.8	0.2	0.0829±0.0773

Table 3: Tuning of the GA parameters; approximation errors for various GA parameters; using two synthetic data curves and one biomedical spectrum

Data	No. of segments	Method	Error magnitude
Smooth	10	Sorted indices	0.0593±0.0434
		Increments	0.0597±0.0372
	20	Sorted indices	0.0234±0.0173
		Increments	0.0233±0.0176
AL spec#1	10	Sorted indices	0.0776±0.0794
		Increments	0.0827±0.0783
	20	Sorted indices	0.0430±0.0557
		Increments	0.0427±0.0578

Table 4: Comparison of two GA approaches; approximation errors for two chromosome representations; using two examples of data curves

Based on the results summarized in Tab. 3 and 4, it seems that set of parameters can be determined as a solid basis to work with in this application. The GA was tested in the rest of the experiments using the following parameters.

- Number of generations = 50
- Population size = 100
- Crossover probability = 0.8
- Mutation probability = 0.1

As for the chromosome representation method, it appears as if neither method has a clear advantage over the other. The method chosen (somewhat arbitrarily) for the continuation of this study was "sorted indices".

Analogously to the figures visualizing the approximations of the bottom-up segmentation, Figure 14 consists of nine plots of the resulting approximation reached by using 2-10 linear segments, when GA-based optimization is employed. Again, the upper-left plot shows the approximation with two lines etc. and the lower-right graph illustrates the result with 10 linear segments.

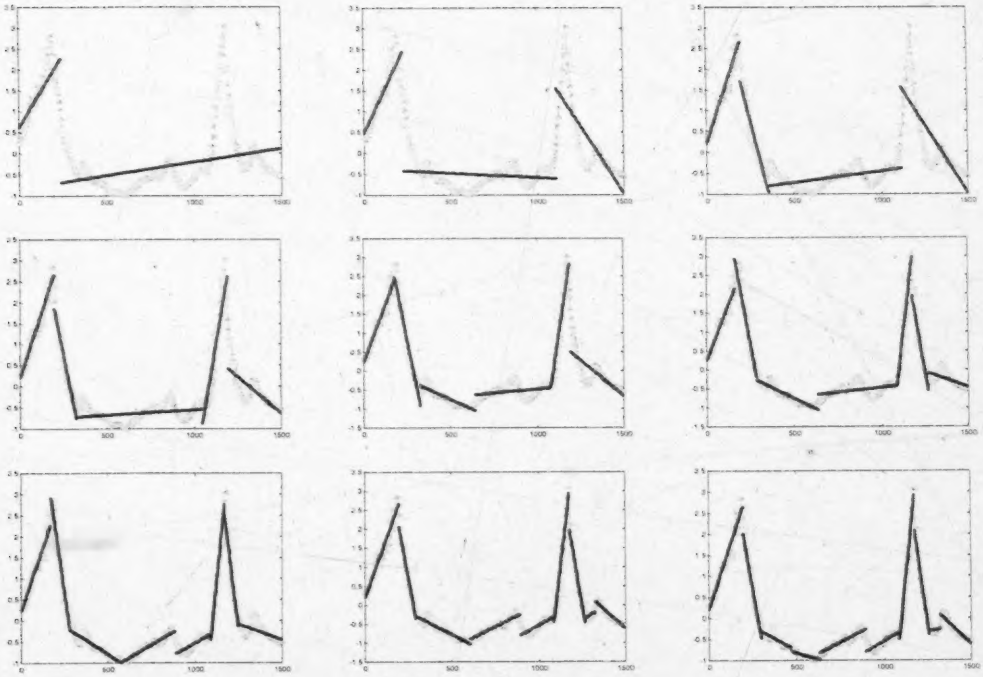


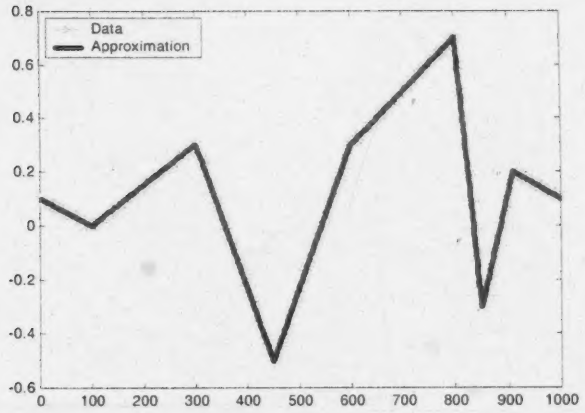
Figure 14: Nine approximations using GA-based optimization with different number of segments; illustrated on AL data spectrum example

4.3.2 Application to artificial data

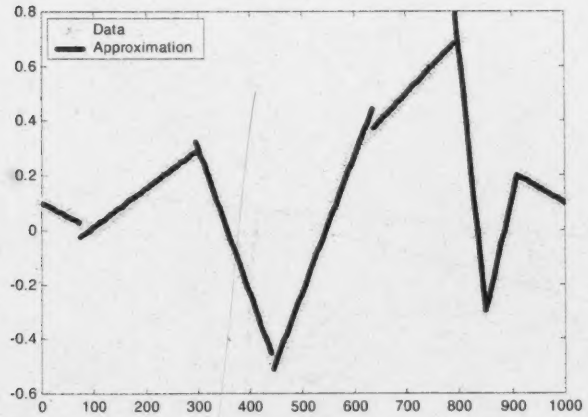
In this section the performance of the GA-optimized linear piecewise approximation is compared to the approximation quality achieved using the bottom-up segmentation method, using various artificial data. The approximation quality evaluation was done as described in the following. For each curve the absolute value of the approximation error is measured for each point in the curve, followed by computing the average and standard deviation of the error magnitude per curve.

Piecewise linear function

The application of the two discussed algorithms is demonstrated in Figure 15 for a piecewise linear function consisting of eight linear segments.



(a)

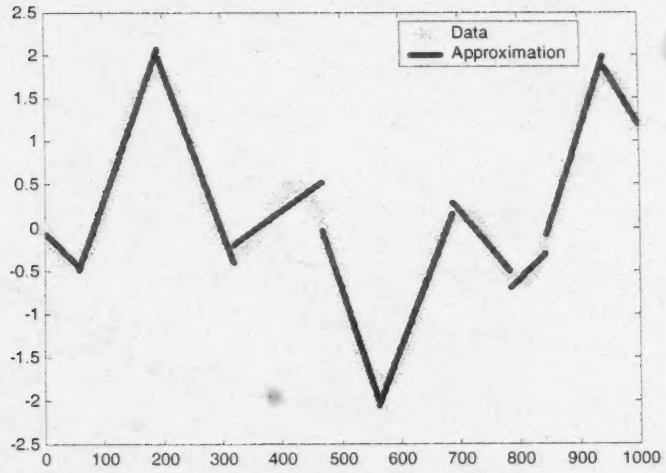


(b)

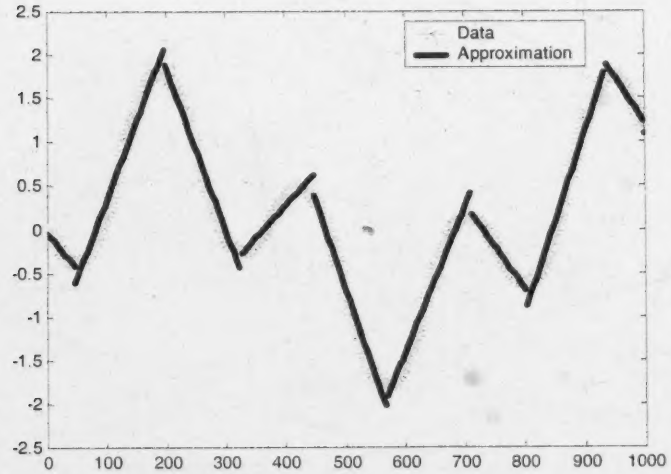
Figure 15: Data and approximation of piecewise linear function using 8 linear segments; bottom-up segmentation (a), GA-optimized (b)

Smooth function

The application of the two discussed algorithms is demonstrated on a smooth function, with additional noise at various levels and using different numbers of linear segments. Figure 16 shows the noiseless curve with the resulting approximation. Figure 17 displays an example of the data after additional noise was applied along with the piecewise linear approximation.



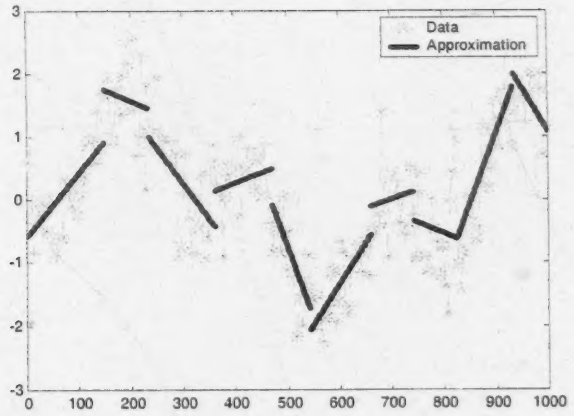
(a)



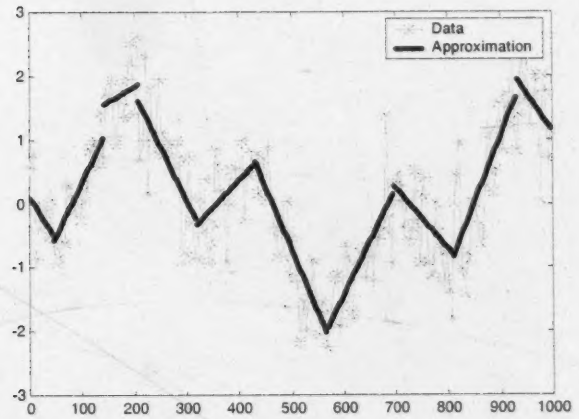
(b)

Figure 16: Data and approximation of a smooth function using 10 linear segments; bottom-up segmentation (a), GA-optimized (b)

Smooth function with noise



(a)



(b)

Figure 17: Data and approximation of a smooth function with noise (noise standard deviation = 0.5) using 10 linear segments; bottom-up segmentation (a), GA-optimized (b)

The numerical results, which consist of the averages and standard deviations of the approximation error (in magnitude), are summarized in Table 5. These include the approximation error for all four cases. The piecewise linear function is approximated using eight segments. The purpose is to test each method's accuracy with the actual number of segments that make up the curve. The three other types of signals are approximated using 10 and 20 segments with each method.

No. of segments	Data	Bottom-up segmentation	GA-based Line fit
8	Piecewise linear	2e-15±3e-15	0.0063± 0.0108
10	Smooth function	0.0721±0.0580	0.0593±0.0434
20		0.0226±0.0218	0.0234±0.0173
10	Smooth w/noise(0.1)	0.1095±0.0829	0.0961±0.0733
20		0.0845±0.0637	0.0810±0.0591
10	Smooth w/noise(0.5)	0.3972±0.3037	0.3973±0.2959
20		0.3908±0.2931	0.3812±0.2912
10	Smooth w/noise(1.0)	0.8406±0.6154	0.8287±0.5999
20		0.8258±0.5886	0.8243±0.5869

Table 5: Piecewise curve approximation: error magnitude average (± standard deviation); comparison of two piecewise linear methods; using artificial data

It can be seen that for the piecewise linear function the bottom-up segmentation method has a significant advantage over the GA-based method. In other cases, the GA-based algorithm is either comparable or at times seems to be slightly better than the bottom-up segmentation method.

4.3.3 Piecewise approximation of a single curve

This section compares the approximation performance of the two methods – bottom-up segmentation, and GA-optimized using the biomedical spectra. The results were produced in the following manner. From each class five spectra were arbitrarily picked. Using each of the methods an approximation was generated for each spectrum example and the magnitude of the approximation error (absolute value) was calculated, and the mean and standard deviation were produced. Using the values of means and standard deviations from five spectra, an average error and standard deviation per approximation method, per class was finally computed. The results were obtained using various values for the number of segments (5,10,20 and 30) devoted to generating the approximations. The numerical results include also GA-optimized piecewise approximation using polynomials of order 2.

The program used for testing the bottom-up segmentation is an available implementation (Keogh, 2003).

Figure 18 illustrates graphically a typical fitness function curve where the x-axis represents the generation index for spectrum no. 1 from class AL, approximated using 20 segments and GA-optimized.

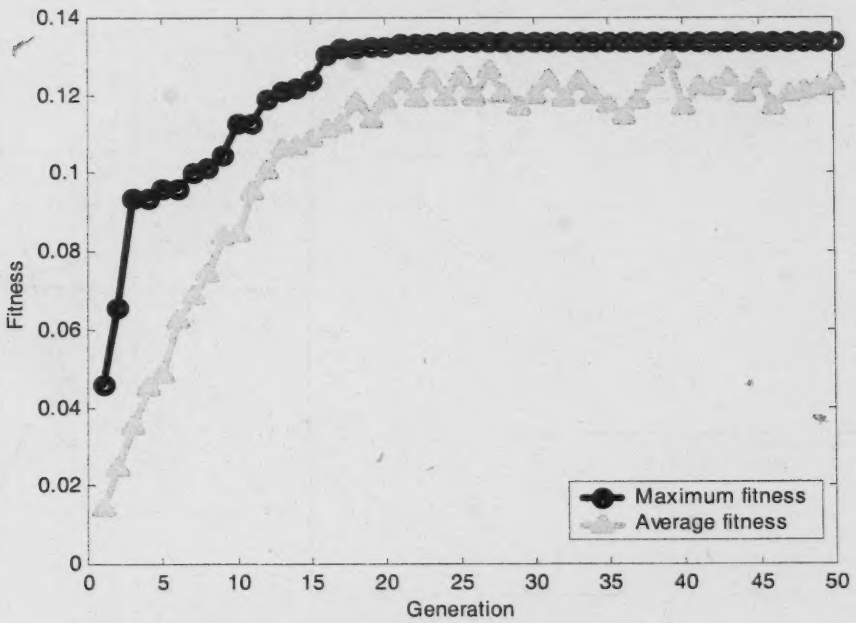
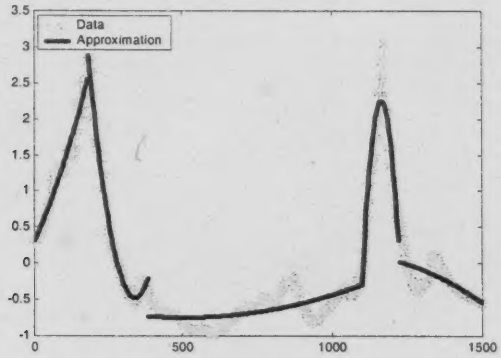
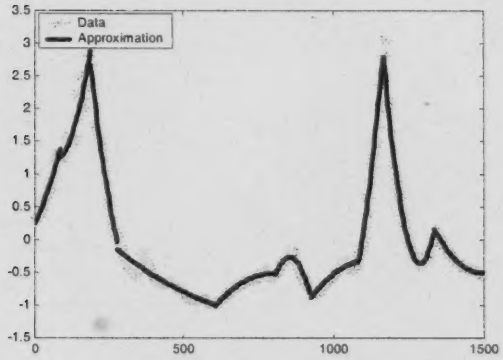


Figure 18: Fitness values (maximum and average) as a function of generation number; spectrum no. 1 from class AL; using 20 segments

The expansion of the GA-based piecewise linear approximation to that of an order 2 polynomial is illustrated in Figure 19. The data used is that of spectrum no. 1 of AL, and the approximation is demonstrated with 5 and 10 segments.



(a)



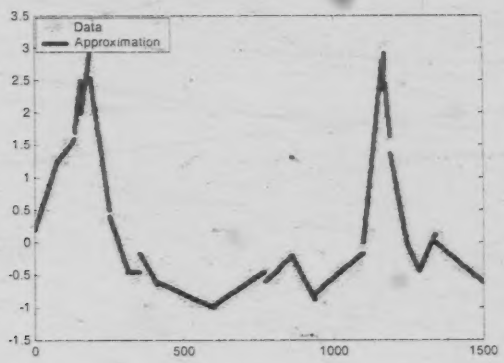
(b)

Figure 19: Data and approximation of AL example no. 1 using GA-optimized method, piecewise polynomial (order 2); 5 segments (a), 10 segments (b)

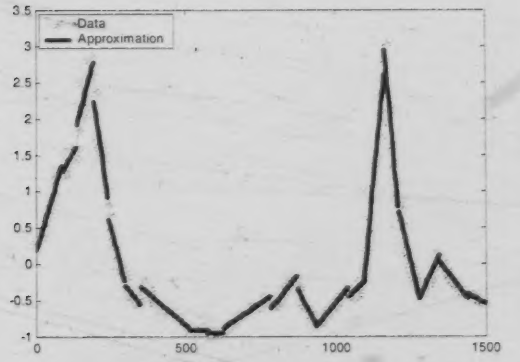
Fig. 20 through 24 demonstrate the nature of the attained approximations using the two discussed algorithms for approximating spectrum no. 1 from each data set, with 20 linear segments. Following the graphic illustrations, numerical results are brought as well, in Table 6. The numerical results consist of the average and standard deviation of the approximation error (in magnitude), for each of the five classes, using each of the approximation methods, using 5, 10, 20 and 30 segments. Performing the approximations for five, arbitrarily picked, spectra from each class and averaging the error-magnitudes average and

standard deviation obtained these values for each case.

AL



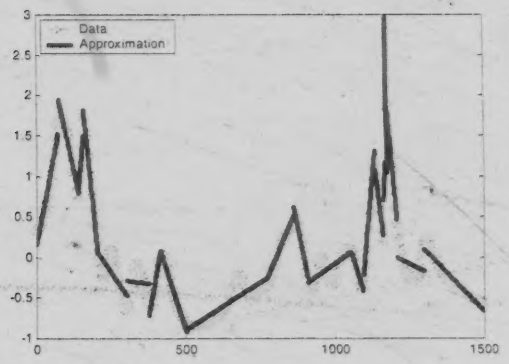
(a)



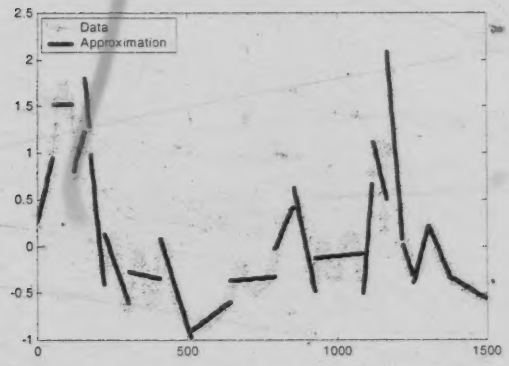
(b)

Figure 20: Data and approximation of AL example no. 1 using 20 linear segments; bottom-up segmentation (a), GA-optimized (b)

GL



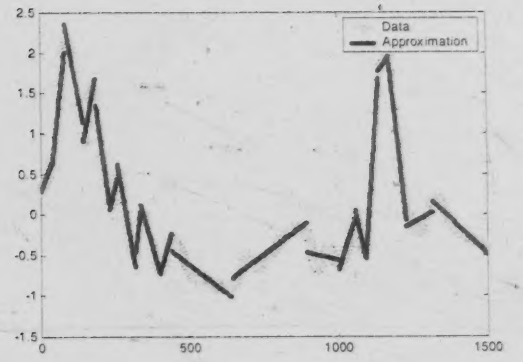
(a)



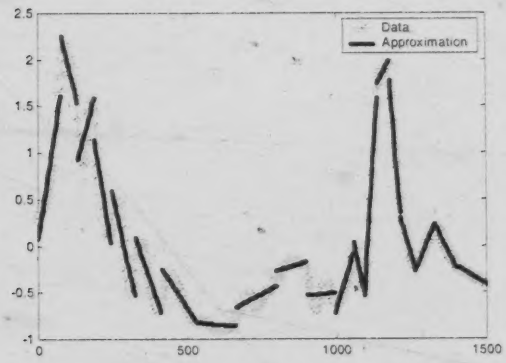
(b)

Figure 21: Data and approximation of GL example no. 1 using 20 linear segments; bottom-up segmentation (a), GA-optimized (b)

KR



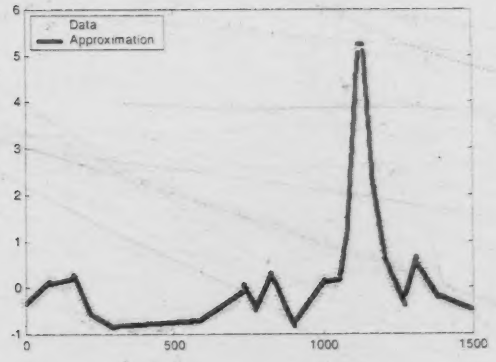
(a)



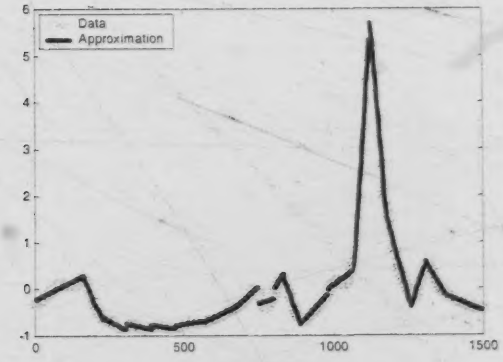
(b)

Figure 22: Data and approximation of KR example no. 1 using 20 linear segments; bottom-up segmentation (a), GA-optimized (b)

PA



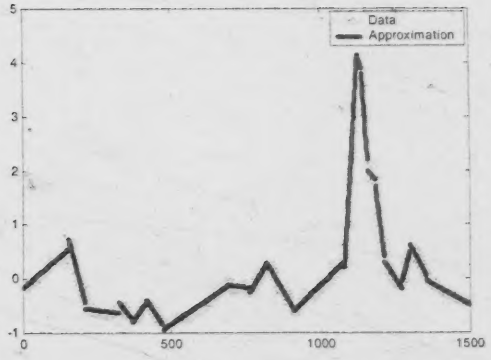
(a)



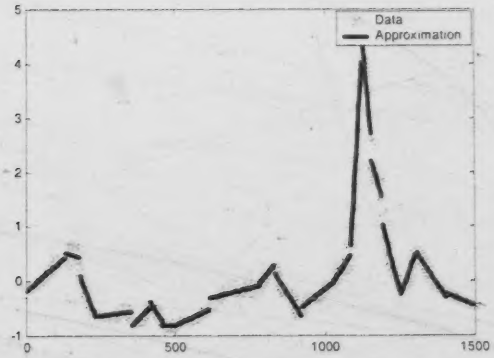
(b)

Figure 23: Data and approximation of PA example no. 1 using 20 linear segments; bottom-up segmentation (a), GA-optimized (b)

TR



(a)



(b)

Figure 24: Data and approximation of TR example no. 1 using 20 linear segments; bottom-up segmentation (a), GA-optimized (b).

No. of segments	Data	Bottom-up segmentation	GA-based Line fit	GA-based Polynomial fit (2)
5	AL	0.202±0.204	0.184±0.177	0.140±0.122
10		0.082±0.088	0.084±0.083	0.053±0.057
20		0.041±0.039	0.047±0.052	0.025±0.034
30		0.023±0.025	0.028±0.039	0.016±0.027
5	GL	0.202±0.190	0.192±0.167	0.154±0.143
10		0.132±0.121	0.127±0.124	0.086±0.084
20		0.062±0.057	0.070±0.079	0.033±0.042
30		0.032±0.030	0.043±0.055	0.020±0.029
5	KR	0.204±0.194	0.185±0.163	0.135±0.127
10		0.117±0.098	0.111±0.100	0.078±0.080
20		0.059±0.052	0.066±0.065	0.033±0.040
30		0.032±0.031	0.041±0.047	0.019±0.031
5	PA	0.199±0.180	0.184±0.161	0.131±0.123
10		0.097±0.096	0.102±0.099	0.060±0.058
20		0.046±0.043	0.051±0.051	0.025±0.032
30		0.025±0.023	0.033±0.038	0.015±0.023
5	TR	0.192±0.164	0.182±0.154	0.128±0.107
10		0.072±0.068	0.080±0.075	0.050±0.049
20		0.035±0.032	0.039±0.041	0.020±0.023
30		0.020±0.019	0.025±0.029	0.010±0.015

Table 6: Piecewise curve approximation: error magnitude average (± standard deviation); comparison of two piecewise linear methods, and one piecewise polynomial (order 2); biomedical spectral data

The results from Table 6 are displayed graphically in Figure 25, for all five data sets and various numbers of segments. The GA-optimized segmentation's performance is comparable to the bottom-up segmentation method, while occasionally achieving improved results, especially when using 5 or 10 segments for the approximation. Note that with a polynomial of order 2 a more accurate approximation is achieved, however, this involves using a more costly representation; since each segment is approximated with a function of three parameters (quadratic function) instead of two (linear function), and this tradeoff has to be considered. In some cases this can lead to undesired consequences due to the *curse of dimensionality* (Bellman, 1961), such as a requirement for a larger number of examples that are not always available.

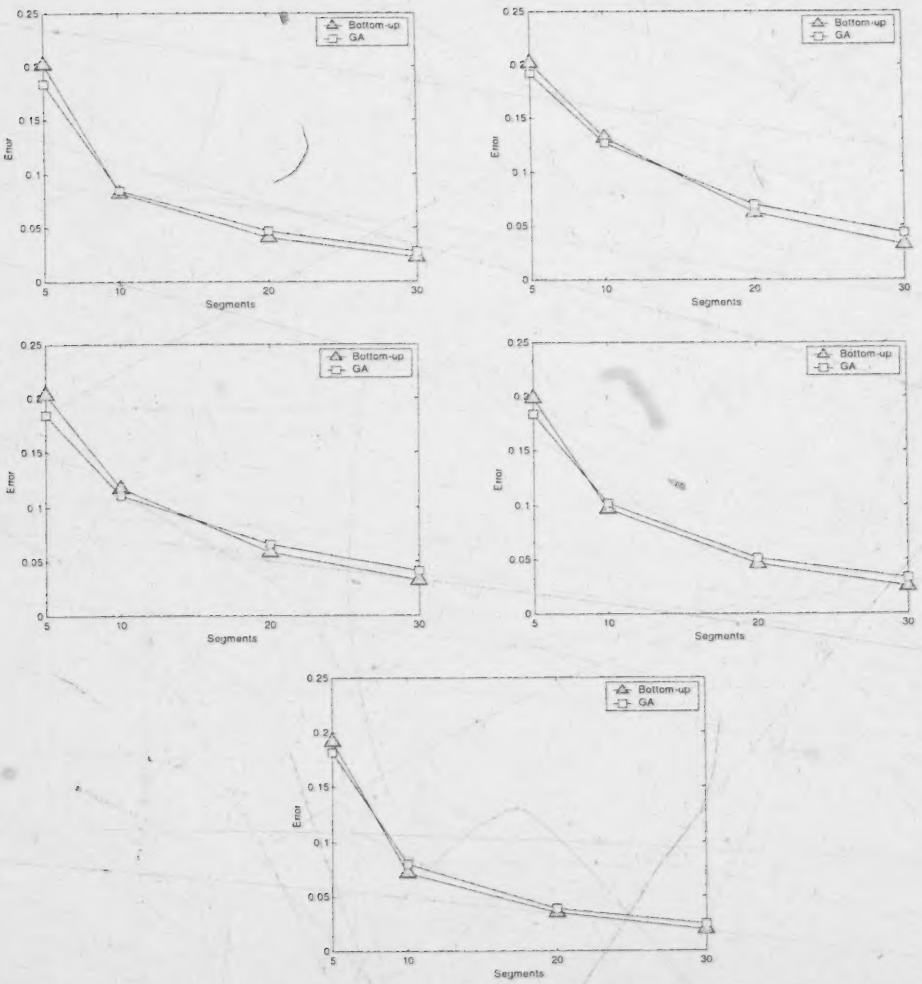


Figure 25: Piecewise linear approximation average errors, of the five data classes using two approximation methods with 5,10, 20 and 30 segments; AL (upper left), GL (upper right), KR (middle row, left), PA (middle row, right), TR (bottom row)

4.3.4 Algorithm expansion: construction of an optimal piecewise approximation for the whole data set

When the motivation behind constructing the approximator is to devise a feature extractor for classification purposes, it would be extremely advantageous if it were possible to achieve an optimal set of breaking points for the piecewise approximation of entire data set. With the GA-based method an expansion is

proposed in order to achieve that purpose. Typically, a classification problem includes a set of examples called a training set that is used for designing the classifier and adjusting its parameters (training phase), and an additional set of examples that are not included in the data for the training phase called testing set, with which the classifier is evaluated. Using the same terminology, the piecewise approximator's set of breaking points should be found in such a way that it would be optimal for the training set, and in a later stage the same set of break points would be used for calculating the quality of the approximation on the testing set, without modifying the breaking points. Note that at this point the testing set has not yet been used for evaluating classification performance, but solely for estimating the piecewise approximator.

In order to expand the capability of the GA-based piecewise approximator, the algorithm underwent the following adjustment. In each generation of the GA, the fitness is calculated using the approximation error calculated with a specific and common set of breaking point, obtained for samples of spectra from the entire training set, contrary to using just one spectrum curve as was done in the single spectrum approximation.

Note that since using all training samples at each generation can be very time consuming, from the computational point of view, another approach is devised and evaluated.

This alternative method consists of randomly selecting a small portion (as small as 10 percent) from the training set (stratified), at each generation, instead of using all samples of the training set. Next, after the GA has reached a solution (meaning an optimal set of breaking points) for the training set it is evaluated using the testing set, by, as done with the training set, calculating the piecewise approximation error for each curve (average and standard deviation), and averaging over all curves in the testing set, generating one quality measure for the whole testing set.

Table 7 summarizes the approximation errors achieved using 5, 10, 20 and 30 segments, and compares the performance of two the approaches: using all training set samples at each generation of the GA vs. employing only 10% (randomly chosen) examples at each generation.

Number of segments	Data	Percentage of training samples per generation	Training set	Testing set
5	AL vs. GL	100	0.2157±0.2084	0.2106±0.2031
5		10	0.2122±0.2150	0.2060±0.2084
10		100	0.1227±0.1418	0.1231±0.1458
10		10	0.1242±0.1315	0.1222±0.1296
20		100	0.0667±0.0771	0.0670±0.0785
20		10	0.0660±0.0816	0.0665±0.0822
30		100	0.0449±0.0670	0.0454±0.0676
30		10	0.0462±0.0669	0.0468±0.0659
5	AL vs. KR	100	0.2029±0.2050	0.1989±0.2037
5		10	0.2141±0.2043	0.2187±0.2121
10		100	0.1148±0.1164	0.1092±0.1101
10		10	0.1146±0.1164	0.1097±0.1105
20		100	0.0686±0.0775	0.0652±0.0723
20		10	0.0724±0.0877	0.0686±0.0820
30		100	0.0448±0.0657	0.0431±0.0626
30		10	0.0415±0.0579	0.0397±0.0525
5	AL vs. PA	100	0.2049±0.2209	0.2024±0.2182
5		10	0.2007±0.2190	0.1969±0.2160
10		100	0.0985±0.1015	0.0975±0.1020
10		10	0.0996±0.1057	0.0982±0.1061
20		100	0.0564±0.0664	0.0569±0.0682
20		10	0.0547±0.0667	0.0539±0.0676
30		100	0.0384±0.0547	0.0390±0.0571
30		10	0.0372±0.0537	0.0376±0.0545
5	AL vs. TR	100	0.1908±0.2060	0.1938±0.2153
5		10	0.2006±0.2152	0.2047±0.2240
10		100	0.0918±0.0942	0.0938±0.0979
10		10	0.0895±0.0919	0.0923±0.0957
20		100	0.0501±0.0593	0.0511±0.0590
20		10	0.0508±0.0637	0.0526±0.0647
30		100	0.0327±0.0471	0.0334±0.0452
30		10	0.0345±0.0497	0.0348±0.0479

Table 7: Piecewise curve approximation of whole data sets: error average (± standard deviation); using 5, 10 ,20 and 30 segments; 100% vs. 10% of examples used per GA generation

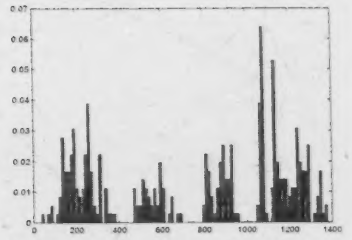
From summarizing the results in Table 7, the following Table 8, is obtained, which shows the average error over all four data sets and for all values used for number of segments, separately for the two cases, i.e. when all training set examples were used at each generation, and when only 10% were used. It is quite evident that using only 10% of the training set examples does not lead to any significant decrease in the

approximation quality, while bringing on very significant reduction in computation time.

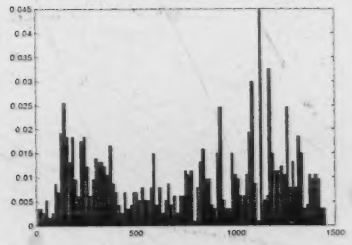
Percentage of training samples per generation	Training set average error	Testing set average error
100	0.1028	0.1019
10	0.1037	0.1031

Table 8: Average approximation errors; using all data set, with 5, 10, 20 and 30 segments; 100% vs. 10% of examples used per GA generation

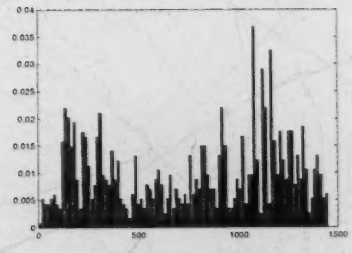
It is interesting to observe, in Figure 26, the distribution of the resulting breaking points, for the case of 10 segments, when letting the GA run separately on all examples of each of the classification problems. In contrast to the method previously employed, by which the GA-based method was used in order to reach a set of breaking points common to all spectra, it is obvious that in this case there is very significant variation of the breaking points among the spectra in each case.



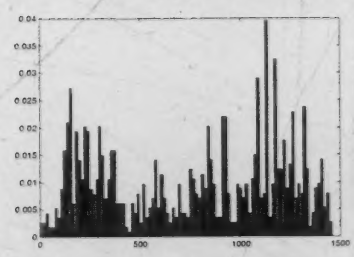
(a)



(b)



(c)



(d)

Figure 26: Distribution of breaking points; AL vs GL (a), AL vs KR (b), AL vs PA (c), AL vs TR (d).

4.4 Classification

Subsequent to the approximation process the incorporation of a classifier as the next stage is examined. For the purpose of evaluation of classification capability, the pseudo-inverse classifier (Duda *et al.*, 2001) is

applied to the extracted features obtained both using the GA-based piecewise approximation and by PCA (Duda *et al.*, 2001).

The pseudo-inverse method

This method is one of several (Duda *et al.*, 2001) tools of computing a linear classifier when such a classifier is desired. The idea at the basis of this classifier is as follows. Define Y as the n-by-(d+1) data matrix, where "n" is the total number of features vectors and d is the data dimension (the number of the matrix columns is "d+1" though due to additional bias). Define also a vector **b** of length n that stands for the target vector of the classifier. The problem is to find a weight vector **w** such that

$$\mathbf{Yw} = \mathbf{b} \tag{3}$$

$$b_i,\ w_j,\ y_{i,j} \in \mathcal{R},\ i=1..n,\ j=1..d+1$$

Since Y is rectangular, the minimization of the classifier's square error

$$\|\mathbf{Yw} - \mathbf{b}\|^2 \tag{4}$$

is achieved using

$$\mathbf{Y}^+ = (\mathbf{Y}^T\mathbf{Y})^{-1}\mathbf{Y}^T \tag{5}$$

which is known as the pseudo-inverse matrix, used to calculate the required weight vector (classifier) by

$$\mathbf{w} = \mathbf{Y}^+\mathbf{b} \tag{6}$$

4.4.1 Using the GA-based method extracted parameters

For the case where the features are extracted using piecewise approximation parameters, a piecewise polynomial approximation of order p can be viewed also as a feature extraction method that performs potentially very significant feature reduction from the original dimension to a dimension of (p+1)m, where m is the number of segments with which the approximation is computed. Table 9 summarizes the classification results for different numbers of segments used to approximate the biomedical spectra. Again, the classification results (using GA-based segmentation) are compared for two cases: (a) all training set examples are used at each generation; (b) only 10% are used at each generation.

Number of segments	Data	Percentage of training samples per generation	Training set	Testing set
5	AL vs. GL	100	98.33	98.67
5		10	99.17	98.67
10		100	100	98.67
10		10	100	100
20		100	100	100
20		10	100	100
30		100	100	100
30		10	100	100
5	AL vs. KR	100	99.07	97.40
5		10	98.15	97.40
10		100	100	98.70
10		10	100	98.70
20		100	100	98.70
20		10	100	98.70
30		100	100	100
30		10	100	98.70
5	AL vs. PA	100	91.94	90.41
5		10	87.90	90.41
10		100	95.97	91.78
10		10	95.16	94.52
20		100	98.39	97.26
20		10	100	97.26
30		100	100	97.26
30		10	100	95.89
5	AL vs. TR	100	86.96	89.87
5		10	90.22	91.14
10		100	96.74	97.47
10		10	98.91	98.73
20		100	100	97.47
20		10	100	98.73
30		100	100	97.47
30		10	100	97.47

Table 9: Correct classification rates using the piecewise curve approximation parameters

Obviously, from the classification results, in Table 9, using only 10% of the training set examples does not lead to any significant decrease in performance, as well as in the approximation quality seen previously.

4.4.2 Using principal component analysis for feature extraction

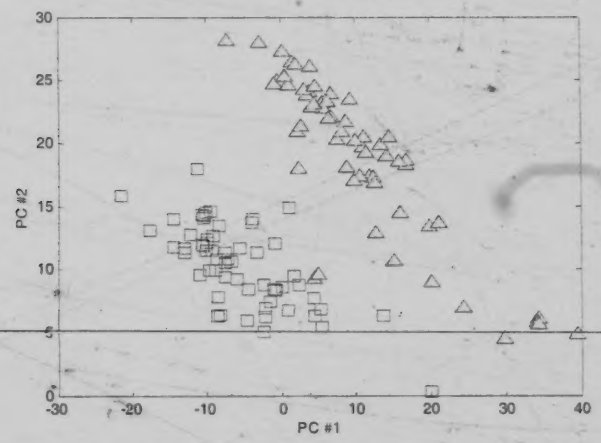
PCA

Principal Component Analysis, also known as the Karhunen-Loeve transform (KLT), is a widely used method for feature extraction used for representing data in a low-dimensional subspace (Duda *et al.*, 2001; Jolliffe, 1986). This technique produces a d-by-n projection matrix, where "n" is the original dimension of the feature space and "d" is the dimension of the reduced feature space. The d rows of the projection matrix are the d eigenvectors related to the d largest eigenvalues of the data's covariance matrix. These eigenvectors are the principal axes of the data set. These eigenvectors are orthogonal and therefore useful as a set of vectors for representing feature vectors that the data consists of.

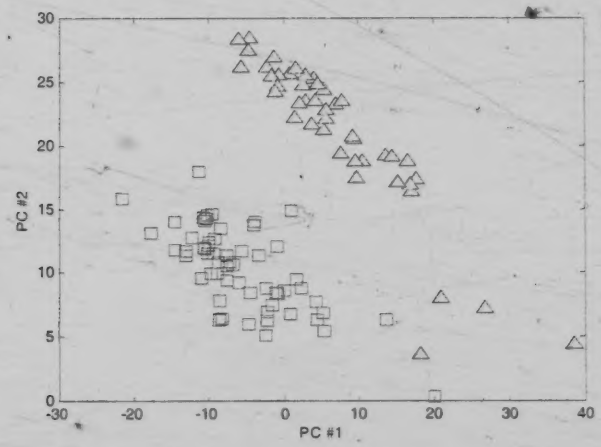
This method of dimensionality reduction finds a linear transformation from the n-dimensional input space to d-dimensional subspace spanned by d orthogonal vectors. The transformation result of the data using PCA is known as the principal components (PCs). The first PC is defined as the direction along which the variance of the input data is maximal and so on until the last PC is the direction of minimum variance thus, PCA creates a new feature space in which the focus of attention can be just relatively few PCs that account for a large amount of the variance in the data.

Following in Table 10 are classification results for each of the four classification problems for different numbers of PCs that are used as inputs for the pseudo-inverse classifier. For illustration purposes, Figure 27 through 30 display the two first PCs, obtained from the training sets and testing sets of the four classification problems: AL vs. GL, AL vs. KR, AL vs. PA, and AL vs. TR. It is apparent from these figures that when using the first two PCs, the AL vs. PA is significantly more difficult than the other three classification problems.

AL vs. GL graphical representation of the first two principal components:



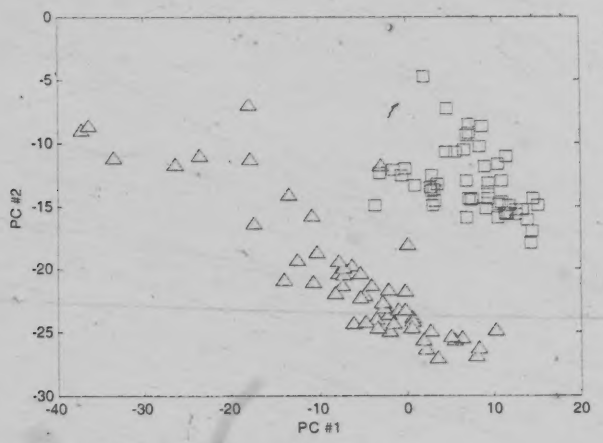
(a)



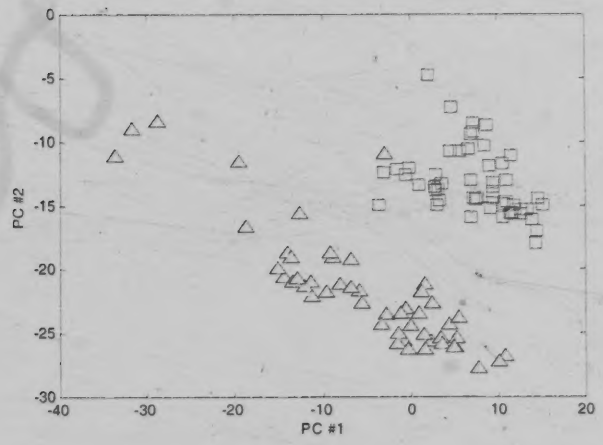
(b)

Figure 27: First two principal components; AL vs. GL data set; training set (a), testing set (b)

AL vs. KR graphical representation of the first two principal components:



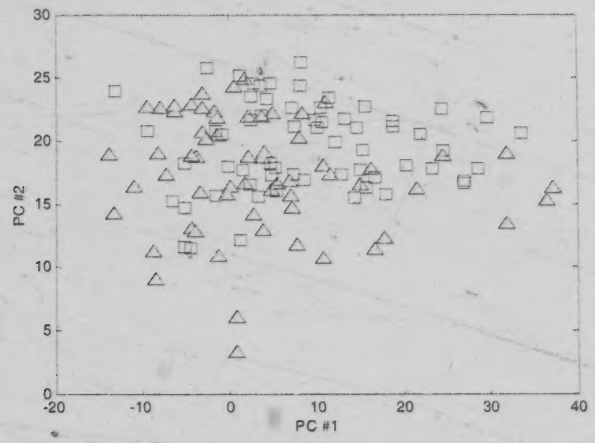
(a)



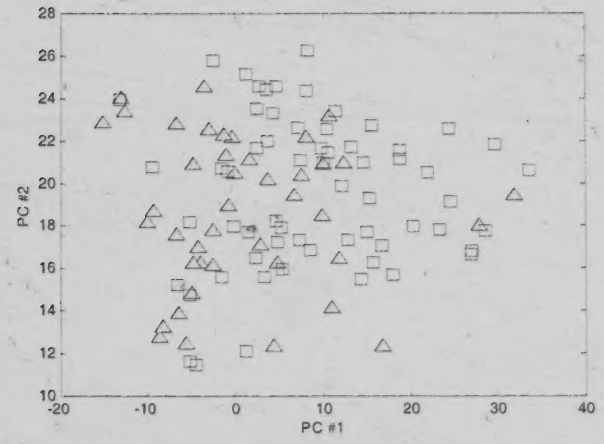
(b)

Figure 28: First two principal components; AL vs. KR data set; training set (a), testing set (b)

AL vs. PA graphical representation of the first two principal components:



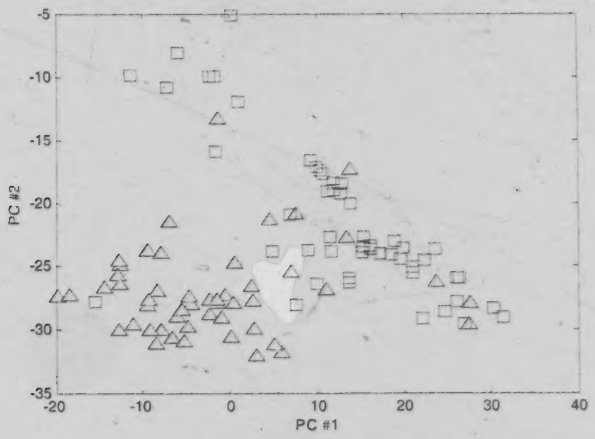
(a)



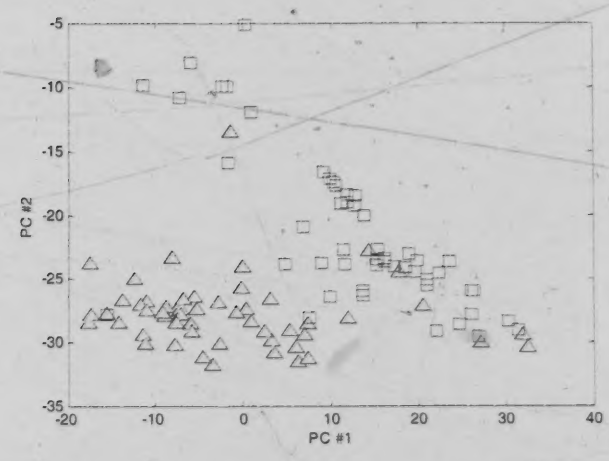
(b)

Figure 29: First two principal components; AL vs. PA data set; training set (a), testing set (b)

AL vs. TR graphical representation of the first two principal components:



(a)



(b)

Figure 30: First two principal components; AL vs. TR data set; training set (a), testing set (b)

No. of PCs	Data	Training set	Testing set
2	AL vs. GL	98.3	98.7
5		99.2	100.0
10		100.0	100.0
20		99.2	97.3
40		100.0	100.0
60		100.0	98.7
2	AL vs. KR	97.22	97.40
5		99.07	98.70
10		99.07	98.70
20		99.07	98.70
40		100	98.70
60		100	98.70
2	AL vs. PA	62.90	61.64
5		90.32	90.41
10		95.97	93.15
20		98.39	93.15
40		100	97.26
60		100	95.89
2	AL vs. TR	89.00	89.87
5		93.00	93.67
10		94.00	92.41
20		97.00	97.47
40		100	97.47
60		100	97.47

Table 10: Correct classification rates for the four classification problems; using PCA with different numbers of PCs

4.5 Summary

This chapter presented the principles and details of a GA-based algorithm for creating a piecewise polynomial representation of data curves. The method was demonstrated using synthetic data and biomedical data, and was shown to perform equally as well and at times better than another widely used method. Finally, it was shown how this method could serve as a feature extractor for a classifier.

5. Fuzzy adaptive logic networks

5.1 Introduction

A comprehensive study of the fuzzy adaptive logic network (FALN) is presented in this chapter. The concept of synergy between *geometry* and *logic*, realized by the FALN, is discussed. The FALN structure, initialization and learning algorithm formulas are detailed, and illustrative examples using synthetic data are offered. Finally, the results of a comprehensive experimental study of the FALN as a classifier are provided.

5.2 Background

The synergy of fuzzy sets and neural network is a multi-faceted development. The commonly accepted point of view concerns the coherent treatment of data and knowledge processing: neural networks concentrate on detailed numeric processing (Hassoun, 1995; Haykin, 1994) whereas fuzzy sets focus on granular computing, thereby being positioned at the logic end of processing (Gabrys and Bargiela, 2000; Ishibuchi *et al.* 1995; Pedrycz and Vasilakos, 1999; Pedrycz and Bargiela, 2002; Simpson, 1992; Simpson, 1993; Sudkamp and Hammell, 1998; Zadeh, 1979; Zadeh, 1996; Zadeh, 1997). This point of view has resulted in a vast array of topologies of neuro-fuzzy systems.

There is also another stream of research of fuzzy neuro-computing in which the focus is on the synergy between (fuzzy) logic and geometry. In this combination, geometric constructs are those of a low-end processing character operating on individual numeric entities. Fuzzy logic constructs operate at the higher-end of the processing spectrum. To some extent, this facet of synergy is visible in some existing constructs such as a network of experts (Jacobs *et al.* 1991; Jordan and Jacobs, 1994; Ramurthi and Ghosh, 1996) or adaptive logic networks (Armstrong *et al.*, 1995).

In the developed neurofuzzy architecture, two classic constructs are exploited: a perceptron originating from core studies on neural networks and a fuzzy logic processor (Pedrycz and Gomide, 1998; Pedrycz and Rocha, 1993) rooted in the realm of fuzzy sets and fuzzy logic.

5.3 The network's architecture

The geometry and logic are the two cornerstones of the proposed topology of the network. The geometry facet of the feature space is captured through a collection of perceptrons. It is obvious that the geometry of the perceptron governed by the expression

$$y = f(\sum_{i=1}^{n} w_i x_i + w_0)$$

$$w_0, w_i, x_i \in \mathcal{R}, i=1..n \tag{7}$$

(where "f" is a sigmoid function) revolves around a single hyper-plane. The modifications of the weight vector (connections), **w**, imply a certain position of the corresponding hyper-plane in the feature space. More concisely, we have $y = f(\mathbf{x},\mathbf{w})$ with **w** denoting the connections of the perceptron and $\mathbf{x} = [1\ x_1\ x_2\ ...x_n]^T$. The combinations of the hyper-planes give rise to more general geometric constructs (refer to Figure 31). Usually, these are formed by inserting an additional layer of units in which the results of computing produced at the first layer of the perceptrons become linearly aggregated. While this type of aggregation is commonly realized by multi-layer neural networks, it corresponds to an interesting and straightforward logical realization producing similar aggregation results.

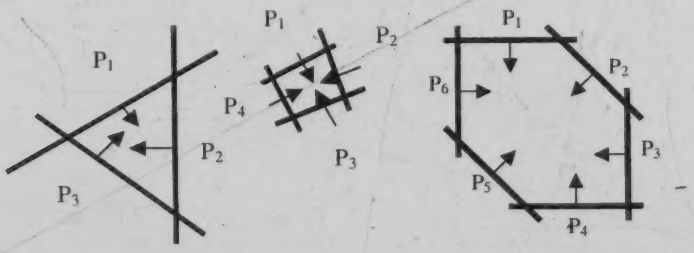


Figure 31: The geometry of perceptrons and their combinations along with their logic interpretations; resulting from building compound predicates; note that a certain class of patterns occupies two disjoint regions in the feature space whose boundaries are approximated by hyper-planes (P_1, P_2, P_3, etc.)

Put differently: the geometric constructs of the perceptrons can be immediately captured and easily quantified in the language of logic. For instance, a convex region delineated by a collection of "c" perceptrons (namely a region of the feature space in which they generate high values of the output) can be represented by the following predicate

$$Q = P_1(x, w_1) \ \& \ P_2(x, w_2) \ \& \ \ldots \ \& \ P_c(x, w_c) \tag{8}$$

This predicate (Q) produces values close to one (meaning it becomes true) when all predicates are true (that is the corresponding perceptrons produce high outputs). In the above expression $P_i(x; w_i)$ denotes the truth-value of the predicate and is equal to the output of this perceptron; $P_i(x; w_i) = f(x, w_i)$. The region formed by the perceptrons comes with an obvious logic interpretation. Usually the geometry of the landscape of the patterns in the feature approximated by the perceptrons is richer than a single convex region. As a matter of fact, a family of disjoint regions may be encountered. To capture this geometry, a union (in set-theoretic sense) of the individual regions described by the Q_is is taken. To articulate the same observation in the language of logic, it is expressed as follows

$$V = Q_1 \ or \ Q_2 \ or \ \ldots \ or \ Q_p \tag{9}$$

where V is a compound *or* predicate formed on a basis of Q_1, Q_2, and P_c. The perceptron governed by (7) produces values in [0,1]. The logic compatible with continuous truth-values exploits fuzzy sets. The above predicates are realized in the form of logic neurons (AND and OR neurons). The two types of compound predicates map directly onto a network of logic neurons, a so-called logic processor cf. (Pedrycz, 1991; Pedrycz, 1997; Pedrycz, 1998; Pedrycz and Gomide, 1998; Pedrycz and Rocha, 1993). The overall network, composed of two functionally distinct substructures, is shown in Figure 32. It will be referred to as a fuzzy adaptive logic network. It is worth stressing that the introductory idea of this nature has been originally introduced by Armstrong (Armstrong *et al.*, 1995); this issue is elaborated on later. The logic aspect and the ensuing processing realized in terms of fuzzy neurons and logic processor have not been a part of the original proposal.

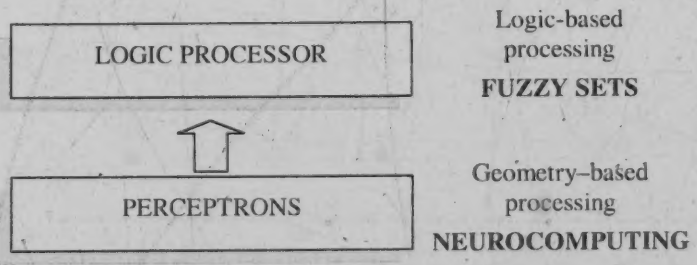


Figure 32: Fuzzy adaptive logic network: a general topology. Note two clearly distinguished subsystems operating in the spaces of features (geometric facet) and logic predicates (logic facet): a collection of perceptrons followed by a logic processor that realizes a logic facet of the classification process

Summarizing, it can be seen that each layer of the above network comes with a clearly defined functionality and a well defined delineated scope of geometric or logic processing.

It is of interest to contrast the proposed paradigm of processing and the resulting architecture with the well-known class of Takagi-Sugeno fuzzy models. Recall that, in essence, a Takagi-Sugeno model is a collection of "local" regression models confined to specific regions of the feature space where these regions are delineated by a collection of fuzzy sets defined in the individual feature spaces. In other words, the model comes as a collection of rules

$$\text{- if feature}_1 \text{ is A and feature}_2 \text{ is B and } \ldots \text{ then } y = a_0 + a_1 x_1 + \ldots + a_n x_n \tag{10}$$

The local models are aggregated by the overlapping fuzzy sets standing in the conditions of the rules. More specifically, the level of activation of each rule μ_i is a weighting factor of the output of the linear expression (y_i) occurring in the conclusion part of the rule. Subsequently, the aggregate $\bar{y}$ is a weighted sum of the form:

$$\bar{y} = \frac{\sum \mu_i y_i}{\sum y_i} \tag{11}$$

The logic part of the model is associated with the feature space (in which a logic aggregate of the fuzzy sets is formed) while the geometry of the construct manifests in the weighted average of the models. In the proposed approach, the aspects of logic and geometry are placed in a reversed order: first, in the feature space the geometry of the perceptrons is considered and then the aggregation comes with a strong logic flavor.

5.3.1 Logic networks – some previous studies

The concept of adaptive logic networks (ALNs) (Armstrong, 1995; Gabrys and Bargiela, 2000) can be viewed as a certain origin of the fuzzy networks studied in this thesis. While some striking differences will become evident later on, the structure itself bears some resemblance. In essence, ALNs are piecewise linear, continuous approximators of non-linear functions that are formed by linear functions (hyper-planes) combined with the use of maximum and minimum operators as illustrated in Figure 33.

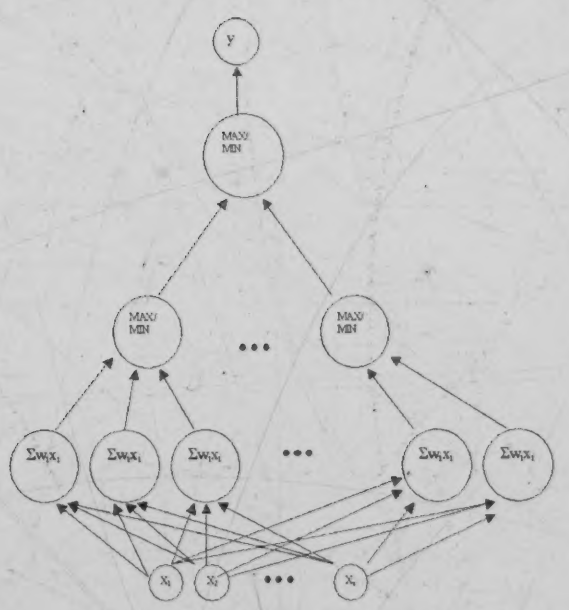
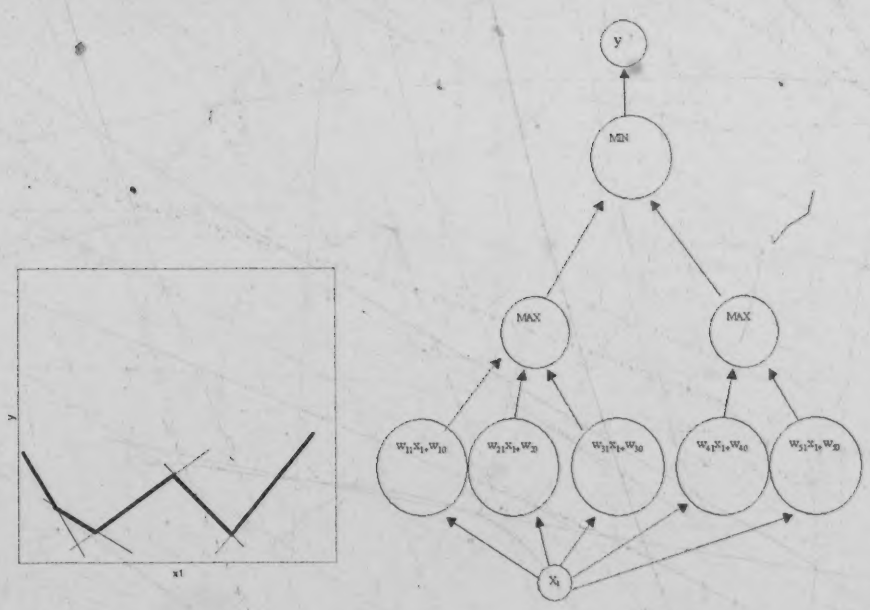


Figure 33: A generic topology of an ALN; note three types of processing unit organized in a layered structure; a weighted sum at the lowest level and a collection of min and max units at the higher levels

To gain a better insight into a way in which the approximation takes place, Figure 34 relates a certain classification problem (more specifically its classification boundary) and its piecewise approximation realized by the corresponding ALN. It is also evident that while the input layer (a collection of the linear functions) is quite uniform, the topology of the max and min nodes is far less structured and in its development substantial structural and parametric learning have to be carried out.



(a) (b)

Figure 34: A piecewise linear boundary (a) and its ALN representation (b)

5.4 Fuzzy neurons and fuzzy logic processing

Proceeding with the logic part of the classifier, it is based on two fundamental classes of logic-driven fuzzy neurons (Pedrycz, 1991; Pedrycz, 1997; Pedrycz, 1998; Pedrycz and Gomide, 1998).

At the basis of the definition of fuzzy neurons lies the concept of triangular norms, which provide standard models for intersection and union operations in fuzzy sets theory. A t-norm is also known as a *triangular norm* and an s-norm is also known as a *triangular co-norm*. Both t-norm and s-norm have to conform to the rules of commutativity, associativity and monotonicity and in that sense they are similar. Additionally they must also satisfy some boundary conditions requirements and in that they differ. The boundary conditions for t-norm are defined as

$$0 \, t \, x = 0, 1 \, t \, x = x \tag{12}$$

whereas for s-norm the boundary conditions are

$$x \, s \, 0 = x, x \, s \, 1 = 1 \tag{13}$$

Table 11 shows a few examples of t-norms.

operation	Min	Product	Lukasiewicz	Drastic product
a t b	min{a,b}	ab	max{0,(a+b-1)}	a, if b=1 b, if a=1 0, if a,b<1

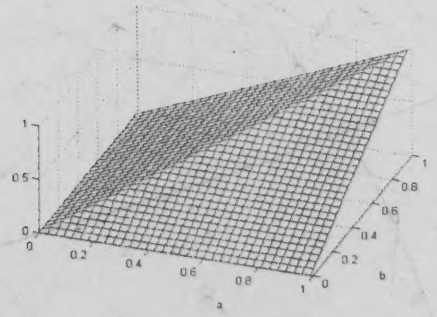
Table 11: t-norm examples

Table 12 shows a few examples of s-norms.

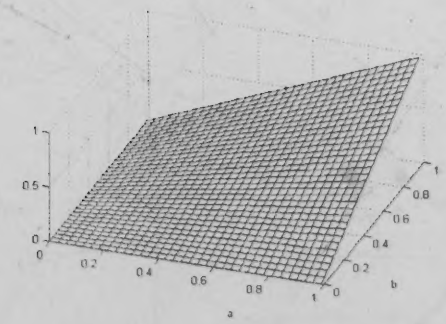
operation	Max	Probabilistic Sum	Lukasiewicz	Drastic sum
a s b	max{a,b}	a+b-ab	min{1,(a+b)}	a, if b=0 b, if a=0 1, if a,b>0

Table 12: s-norm examples

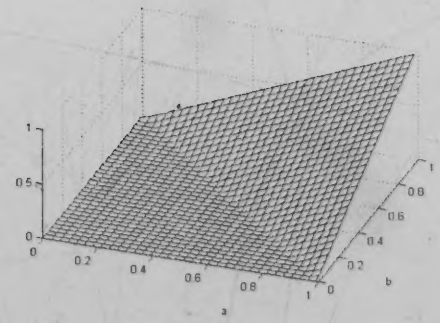
Figure 35 below shows examples of surfaces resulting from t-norm operations.



(a)



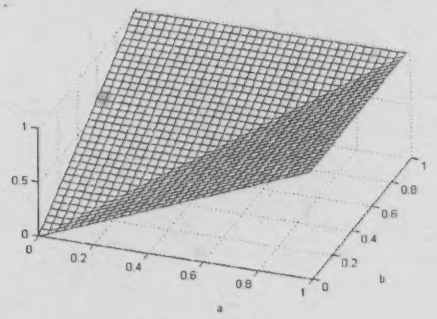
(b)



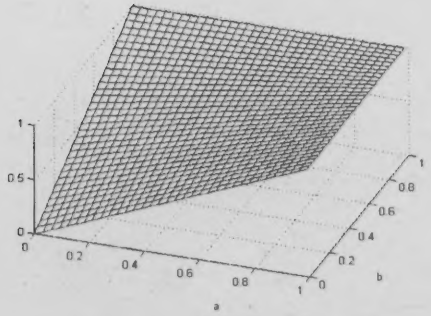
(c)

Figure 35: Examples of t-norms; min (a), product (b), Lukasiewicz (c)

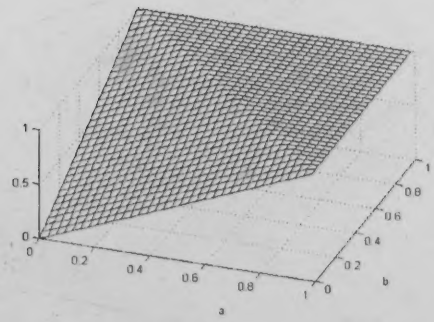
Figure 36 below shows examples of surfaces resulting from s-norm operations.



(a)



(b)



(c)

Figure 36: Examples of s-norms; max (a), probabilistic sum (b), Lukasiewicz (c)

Based on this concept we can proceed to the definitions of the processing elements known as OR neurons and AND neurons as follows.

OR neuron The neuron processes n-inputs $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ (where each input is in the unit interval) and completes an s-t composition operator of the form

$$y = OR(\mathbf{x}, \mathbf{w}) = \overset{n}{\underset{i=1}{S}}(w_i t x_i) \tag{14}$$

where $\mathbf{w} = [w_1\ w_2 \ldots w_n]$ are adjustable weights (connections) confined to the unit interval. Owing to the type of the composition, it is worth noting that if all weights are set to 1 then the output is just a standard OR combination of the inputs (OR gate). The higher the value of the connection, the more relevant the corresponding input. Observe that a commonly used max-min composition is a special case of the operation introduced above. This is illustrated in two dimensions in Figure 37 using the product t-norm and the probabilistic sum s-norm.

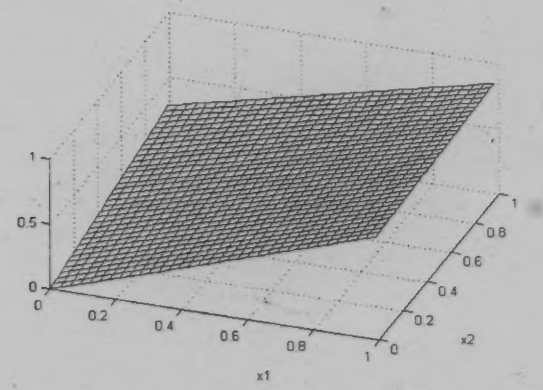


Figure 37: Characteristics of OR neuron; with $w_1 = 0.8$, $w_2 = 0.3$

AND neuron The neuron processes n-inputs $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ (where each input is in the unit interval) and completes an t-s composition operator of the form

$$y = AND(\mathbf{x}, \mathbf{w}) = \overset{n}{\underset{i=1}{T}}(w_i s x_i) \tag{15}$$

If all connections are set to zero, then the result is a standard (non-weighted) combination of the inputs (AND gate).

The connections of the neurons provide us with a significant level of parametric flexibility that helps calibrate how much an individual input (x_i) impacts the output of the neuron. In light of the properties of

the t- and s-norms, the following can be said

- in the case of AND neurons: higher values of the connections imply less impact associated with the input; in limit where $w_i = 1$, this eliminates x_i (x_i is treated as irrelevant)
- in the case of OR neurons: higher values of the connections imply less impact associated with the input; in limit where $w_i = 0$, this eliminates x_i (x_i is treated as meaningless)

By noting that min and max are special cases of triangular norms and co-norms and with the weights set to 0 or 1, the result is the same processing units as encountered in the ALN architectures.

Following is an illustration in Figure 38 using the product t-norm and the probabilistic sum s-norm.

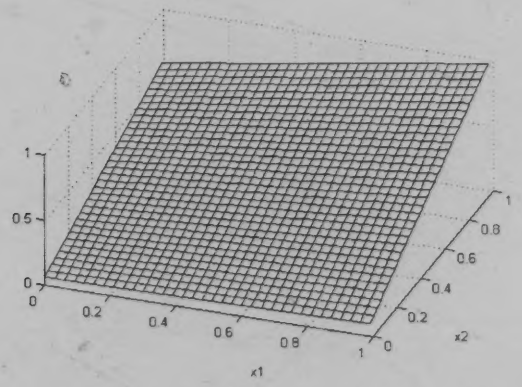


Figure 38: Characteristics of AND neuron; with $w_1 = 0.6$, $w_2 = 0.1$

5.5 Illustrative examples

The goal here is to visualize a diversity of the non-linear characteristics of the network that arises while changing the values of the connections. For illustrative purposes a two-dimensional feature space (x_1-x_2) so that each perceptron has only two input variables is considered. The non-linear element of the perceptron is realized as a standard sigmoid function. The logic processor is reduced to a single AND neuron. This obviously gives rise to the limited geometry of the construct in which a variety of shapes of a single region in the feature space is exhibited in the previous section. In the AND neuron a t-norm is implemented through a product operation while an s-norm is realized as a probabilistic sum. The plots contained in Figure 39 illustrate the characteristics of the network for several selected combinations of the values of the connections of the neuron. In all cases there are three perceptrons with the following linear combinations of the variables: $3x_1 + 5x_2 - 1$, $0.5x_1 - x_2 + 6$, and $-2x_1 + 0.8x_2 - 0.1$ with their underlying geometry visualized in

Figure 40.

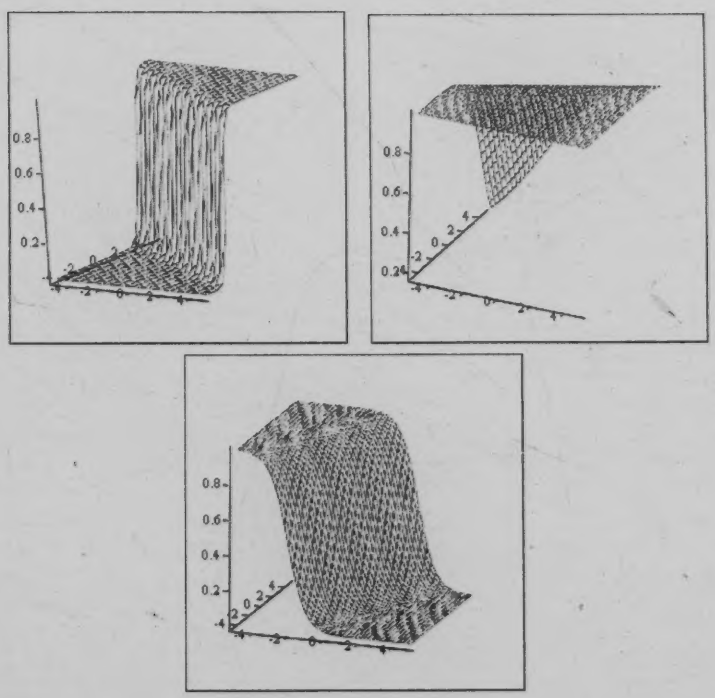


Figure 39: 3D plots of the characteristics of the three perceptrons used in the network

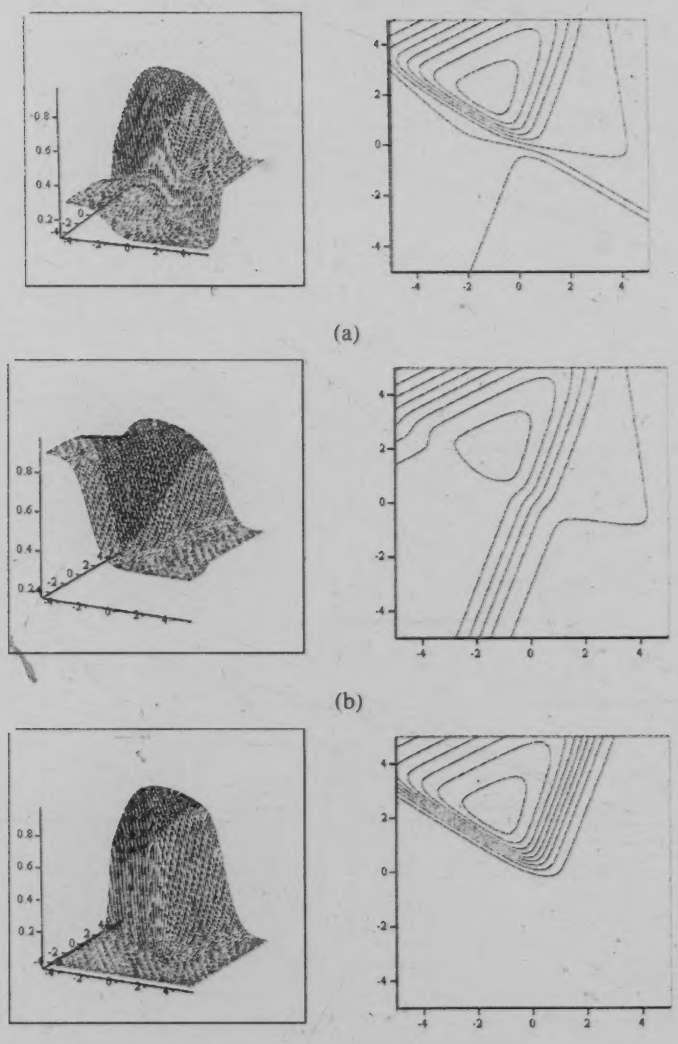


Figure 40: 3D and contour plots of the fuzzy neural network for selected combinations of the connections of the AND neuron: w =[0.3 0.4 0.0] (a) w = [0.9 0.0 0.4] (b) w =[0.0 0.2 0.0] (c) (the last coordinate of the weight vector is a bias)

5.6 Detailed learning scheme

The learning in the network considered here is parametric. It is assumed that the topology of the network

has been given in advance while the parameters (connections) have to be optimized. The performance index Q assumes a standard form of a sum of squared errors between the output of the network and required target, that is

$$Q = \sum_{k=1}^{N} (\mathbf{target}(k) - \mathbf{y}(k))^T (\mathbf{target}(k) - \mathbf{y}(k)) \tag{16}$$

where X symbolizes the training data set formed by pairs of inputs and required outputs (targets) $X = \{\mathbf{x}(k), \mathbf{target}(k)\}$, k=1, 2, ..., N; where "N" is the number of examples in the training data. The updates of the connections (the logic processor as well as the collection of the perceptrons) are guided by the standard gradient-based update scheme that can be symbolically written down as

$$\mathbf{connections}(\text{iter}+1) = \mathbf{connections}(\text{iter}) - \alpha \nabla_{\mathbf{connections}} Q \tag{17}$$

with the term denoted as **connections** that captures all parameters of the network; *iter* denotes consecutive learning epochs and α stands for a positive learning rate.

In what follows, an on-line learning scheme is discussed so that the connections are updated after the presentation of an individual input-output data pair. The input and output are referred to as **x** and target, respectively. The detailed notation is shown in Figure 41; there are "n" inputs to the network (dim (**x**) =n); "p" perceptrons, "hidden" number of the AND nodes of the logic processor and "m" outputs, dim(**y**), dim(**target**) = m. Subsequently, the connections between the consecutive layers of the network are organized in a matrix form, namely U, V, and W where the dimensions of all these matrices correspond with the dimensions of the respective layers.

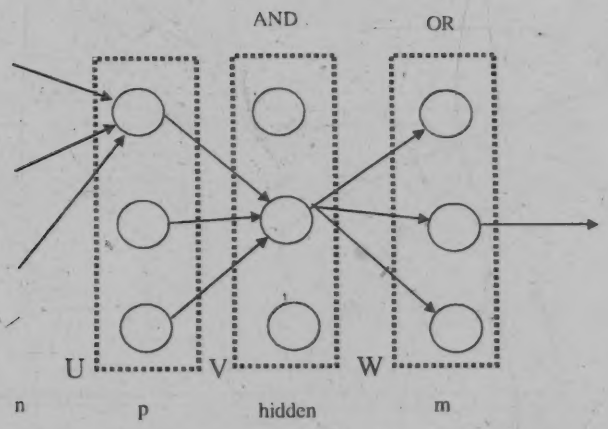


Figure 41: The detailed topology of the network; shown are all connections and sizes of the layers

The crux of the gradient-based learning is the gradient of Q. In Table 13 follows a summary of the detailed formulas for the gradient of Q computed with respect to the connections of the neurons. Note that these expressions can be derived once we confine ourselves to the specific form of the triangular norms and co-norms. In what follows product and probabilistic sum are considered as the models of the logic operators. Their differentiability allows precise mathematical implementation of gradient-based learning. The sigmoid non-linearity used in the neurons of the perceptron is of the form $1/(1+\exp(-u))$.

The notations for Table 13 are as follows:

- y_i is the output of OR neuron i
- g_j is the output of AND neuron j
- z_k is the output of perceptron k
- The weights U, V and W are as depicted in Figure 41, above

$$\frac{\partial Q}{\partial w_{js}} = -2(\text{target}_j - y_j)\frac{\partial y_j}{\partial w_{js}} = -2(\text{target}_j - y_j)g_s(1 - A_{js})$$

where

$$A_{js} = \sum_{\substack{i=1 \\ i \neq s}}^{\text{hidden}} (g_i tw_{ji})$$

$$\frac{\partial Q}{\partial v_{st}} = -2\sum_{j=1}^{m} (\text{target}_j - y_j)\frac{\partial y_j}{\partial v_{st}}$$

with the following expressions for the above derivative

$$\frac{\partial y_j}{\partial v_{st}} = \frac{\partial y_j}{\partial g_s}\frac{\partial g_s}{\partial v_{st}} \qquad \frac{\partial y_j}{\partial g_s} = w_{js}(1 - A_{js}) \qquad \frac{\partial g_s}{\partial v_{st}} = B_{st}(1 - z_t)$$

and

$$B_{st} = \prod_{\substack{j=1 \\ j \neq t}}^{p} (z_j sv_{sj})$$

$$\frac{\partial Q}{\partial u_{ik}} = -2\sum_{j=1}^{m} (\text{target}_j - y_j)\frac{\partial y_j}{\partial u_{ik}}$$

where

$$\frac{\partial y_j}{\partial u_{ik}} = \sum_{l=1}^{\text{hidden}} \frac{\partial y_j}{\partial g_l}\frac{\partial g_l}{\partial z_i}\frac{\partial z_i}{\partial u_{ik}}$$

and

$$\frac{\partial y_j}{\partial g_l} = w_{jl}(1 - A_{jl}) \qquad \frac{\partial g_l}{\partial z_i} = B_{li}(1 - v_{li}) \qquad \frac{\partial z_i}{\partial u_{ik}} = z_i(1 - z_i)x_k$$

Table 13: The detailed formulas for the update of the connections of the FALN

For the sigmoid function equipped with the steepness parameter (σ), that is

$$z = f(u) = \frac{1}{1 + e^{-\sigma u}} \tag{18}$$

the corresponding gradient-based learning scheme is

$$\frac{\partial Q}{\partial \sigma_i} = \frac{\partial Q}{\partial z_i}\frac{\partial z_i}{\partial \sigma_i} \tag{19}$$

More specifically,

$$\frac{\partial z_i}{\partial \sigma_i} = \frac{\partial}{\partial \sigma_i}\left(\frac{1}{1+e^{-\sigma_i u_i}}\right) = \frac{1}{(1+e^{-\sigma_i u_i})^2} u_i e^{-\sigma_i u_i} = z_i^{\,2} u_i e^{-\sigma_i u_i} \tag{20}$$

that is

$$\frac{\partial Q}{\partial z_i} = -2\sum_{j=1}^{m} (\text{target}_j - y_j)\frac{\partial y_j}{\partial z_i} \tag{21}$$

where

$$\frac{\partial y_j}{\partial z_i} = \sum_{l=1}^{\text{hidden}} \frac{\partial y_j}{\partial g_l}\frac{\partial g_l}{\partial z_i} \tag{22}$$

5.7 Two design issues: structure selection and initialization of the learning

There are two fundamental aspects of the development of fuzzy adaptive logic networks (FALNs), namely a selection of their structures and a suitable initialization of the parameters to be optimized throughout the learning process.

The first task falls under the umbrella of structural optimization and concerns two fundamental components, the number of perceptrons and AND neurons. The selected structure links directly with the geometry of the patterns under classification. This could not be easily envisioned in detail but some general hints are apparent; in particular, two guidelines apply. When dealing with separate disjoint classification regions, individual AND neurons are required to cope with this aspect of the geometry of patterns. On the other hand, to approximate more complex classification regions, more perceptrons are needed. Obviously, we do not have exact numbers but with these two guidelines we may set up a certain topology. In general, it is possible to start with a large network and then start scaling it down by eliminating perceptrons and/or AND neurons. One can also proceed in a bottom-up design mode by starting off with a small network,

optimizing it and, if required, expanding it. In many cases the latter method is the preferred, since a linear classifier (or nearly linear) may perform quite well.

The second issue deals with sound initial values of the parameters of the networks and this concerns the parameters (coefficients) of the perceptrons and the weights of the logic neurons. One can distinguish between several possible alternatives. It is worth stressing that the matter of weight initialization is not special to this type of classifier but becomes common to neural networks in general, several studies can be looked into (Bandyopadhyay and Pal, 1999; Bioch *et al.*, 1995; Duch and Adamczak, 1998; von Schmidt and Klawonn, 2001) for a variety of initialization techniques. The detailed structure of the network can be further attained using various cross-validation techniques. As far as the initialization is concerned, several main avenues are pursued.

Random initialization

This is the simplest scenario in which the assumption is that a random configuration of the connections leads to the highest diversity and in this way unbiased learning can be carried out. While this approach could work well in many cases, it is not ideal and may fail in cases when the configuration of the connections is quite remote from the optimal ones. Obviously the problem is that one never knows such optimal connections as well as it is difficult to quantify how remote they could be from the solution to assure the convergence of the learning algorithm.

Pseudo-inverse initialization

The other approach is to attempt to position the initial values of the parameters (connections) in the region where the approximate position could be. Obviously, one is not able to determine their detailed values but hopefully they are close enough to support successful learning. A natural starting point is to adopt a linear form of the classifier as a preliminary solution to the problem. Under this assumption, a solution suitable for the linear problem can be sought. One such method uses the standard pseudo-inverse (Duda *et al.*, 2001). The advantage of this method is that it produces hyper-planes that cross the classification boundary (or come close to it) while being simple and straightforward to implement and fast to execute. This method might run into numerical problems for data of high dimensionality, but this difficulty can be overcome simply by running it on a small (as required) subset of the data. The initial values might be in that case not as good as the values obtained from a large subset of the data, but these values serve only as initial values for the learning process which should compensate for that. Even though such a linear classifier may not be suitable and lead to poor classification performance, it is a sound starting design point.

Initialization using a linear classification tree

While initialization based on pseudo-inverse offers potentially better initial conditions for learning, there is still the problem of trying to approximate the number of perceptrons and AND-neurons required to handle a given data set for classification. Note that the focus here is on an FALN for which the number of OR-neurons is simply 1 since for a 2-class classification problem 1 output is sufficient, and a classification problem of L classes can always be formulated as L 2-class classification problems.

The use of a linear classification tree offers interesting benefits. In enables us to get some insight as to the complexity of the specific classification problem in terms of the number of network parameters needed to achieve separability.

It is straightforward to derive from the resulting linear classification tree an equivalent FALN that provides both initial weights for the perceptrons, and weights for the AND neurons.

The linear classification tree algorithm used for this purpose (Mangasarian, 1993; Bennet, 1992) is called The Multi-Surface Method Tree (MSM-T).

The basic idea of the MSM-T is: let A and B be finite disjoint point sets in n-dimensional Euclidean space. The goal of MSM-T is to determine a sequence of planes in an n-dimensional Euclidean space that separate the sets A and B as follows:

1. Determine a plane in an n-dimensional Euclidean space that minimizes the average "distances" of misclassified points – using linear programming (LP). A point from set A is misclassified if it lies on the side of the separating plane assigned to B. Similarly, a point from set B is misclassified if it lies on the side of the separating plane assigned to A.
2. If the regions, assigned to A and B, contain only (or to a permitted degree) points of the set A or B, then stop. Otherwise, generate another error-minimizing plane (in 1) in this region.

The sequence of planes generated can be viewed as a decision tree. For each node in the tree, the best split of the points reaching that node is found by solving the linear program in 1, above. The node is split into 2 branches, and the same procedure is applied until there are only (or as predefined) points of one set at the node.

The benefit of using the MSM-T algorithm is that its result is collection of hyperplanes that divide the space into regions each of which containing points of mostly one class. The number of hyperplanes can be related to the number of perceptrons with which to construct the FALN and the hyperplane parameters can be used to as initial values for the learning process. The learning process, along with cross-validation, can be used for optimizing the FALN parameters.

Formulating a classification problem as a linear programming (LP) problem (Duda *et al.*, 2001)

LP is a mathematical tool for solving optimization problems. It is basically a method for minimizing a linear objective function, subject to a set of linear inequalities that express constraints that the solution must conform with.

A typical LP problem can be written as follows.

Minimize the objective function

$$z = \boldsymbol{\alpha}^T \mathbf{u} \tag{23}$$

subject to the constraint

$$\mathbf{D}\mathbf{u} \geq \boldsymbol{\beta} \tag{24}$$

where $\mathbf{u}$ and $\boldsymbol{\alpha}$ is an m-by-1vector, $\boldsymbol{\beta}$ is an k-by-1 vector and D is an k-by-m matrix.

$u_i, \alpha_i \in \mathcal{R}$, i=1..m, $\beta_i \in \mathcal{R}$, i=1..k, $d_{ij} \in \mathcal{R}$, i=1..k, j=1..m

A widely used method for solving LP problems is the simplex algorithm.

For clarifying the application of LP to classification, note that "D" is related to the classification data-matrix. Since LP imposes a restriction

$$\mathbf{u} \geq 0 \tag{25}$$

the relation between $\mathbf{a}$ (the linear classifier's weights) and $\mathbf{u}$ is defined

$$\mathbf{a} = \mathbf{a}^+ - \mathbf{a}^- \tag{26}$$

where

$$\mathbf{a}^+ \equiv \frac{1}{2}(|\mathbf{a}| + \mathbf{a}) \quad , \quad \mathbf{a}^- \equiv \frac{1}{2}(|\mathbf{a}| - \mathbf{a}) \tag{27}$$

since $\mathbf{u}$ is later constructed using $\mathbf{a}^+$ and $\mathbf{a}^-$ this results in a non-negative $\mathbf{u}$.

Now, assuming that the classification problem involves "n" vectors $\mathbf{y}_1, \ldots, \mathbf{y}_n$, and that we need to find a weight vector $\mathbf{a}$ that will satisfy:

$$\mathbf{a}^T \mathbf{y}_i \geq b_i; \quad i = 1..n \tag{28}$$

where $\mathbf{b}$ is the target vector, $\mathbf{b} = [b_1, b_2, \ldots, b_n]$.

This can be formulated as a linear programming problem by introducing an additional variable "t", such that

$$\mathbf{a}^T\mathbf{y}_i + t \geq b_i > 0; \quad i = 1..n$$

(29)

and

$$t \geq 0$$

(30)

For t that is large enough, then we will be able to satisfy all these constraints. As an example, the constraints can be satisfied by choosing a = 0 and $t = \max_i\{bi\}$. Although this is a solution, it isn't useful for solving our classification problem.

Next, the following matrices and vectors are constructed for the LP solver (e.g. simplex):

$$D = \begin{bmatrix} y_1^t & -y_1^t & 1 \\ y_2^t & -y_2^t & 1 \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ y_n^t & -y_n^t & 1 \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} a^+ \\ a^- \\ t \end{bmatrix}, \quad \boldsymbol{\alpha} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \quad \boldsymbol{\beta} = \begin{bmatrix} b_1 \\ b_2 \\ \cdot \\ \cdot \\ \cdot \\ b_n \end{bmatrix}$$

(31)

The LP solving algorithm finds the minimum value for t. The desired solution is that which minimizes t (i.e. t = 0, under the constraint $t \geq 0$). A solution of t = 0 indicates that the feature vectors are linearly separable. Otherwise, i.e. for the case where the vectors are linearly non-separable, the resulting t will be positive.

Illustration of how to obtain an FALN from a linear classification tree

The translation of a linear classification tree to an equivalent FALN is done as follows. The MSM-T algorithm produces a classification tree of the form illustrated in Figure 42.

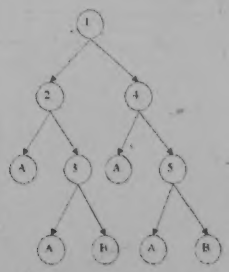


Figure 42: Linear classification tree example

The data set is comprised of samples from two classes A and B. Each of the numbered nodes in the tree represents a linear function of the input features; hence it is called a *linear* classification tree. This linear function represents in fact a hyper-plane at which a decision criterion is evaluated for each feature vector classified at that node. According to the result of this evaluation (determining on which side of the hyper-plane the feature vector lies) the data is classified either to the right branch or to the left branch, until it reaches a leaf-node (highlighted in gray) where it is to be classified as belonging to one of the two classes.

In order to produce an equivalent FALN we first determine for which class (A or B) the FALN should produce 1, while data belonging to the other class should produce 0. Assuming that data belonging to class B should produce 1, then for each of the "B" leaf-nodes, the collection of linear functions leading from the tree root to the leaf-node makes up the perceptrons that serve as inputs to the same AND unit. Note that the sign of linear function parameters has to be considered appropriately at each node (according to whether the criterion at lead to right branch or a left branch). Thus a network (though still not in the form of the fully connected FALN) that is equivalent to the classification tree in Figure 42 can be seen in Figure 43. The numbers of the perceptrons corresponds to the appropriate linear function, and 1* means that the parameters of linear function #1 are to be negated. In order to complete this network to the form of an FALN, all perceptrons need to be connected to all AND neurons and their connection need to be assigned values. This is done by simply assigning a value of 1 to the connection between each of the perceptrons and each AND neurons that are not connected in Figure 31, and a value of 0 otherwise.

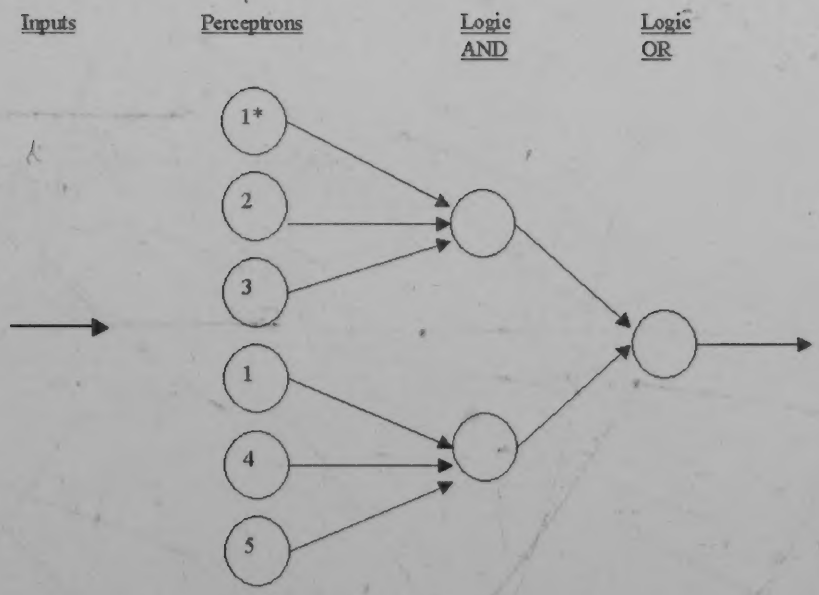


Figure 43: Network equivalent to the linear classification tree

Figure 44 shows the equivalent fully connected FALN structure. As to the values of the connection weights, if V stands for the weights at the AND neurons inputs and W for the weights at the OR neuron input, then we have:

$$V = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

$$W = [1 \quad 1]$$

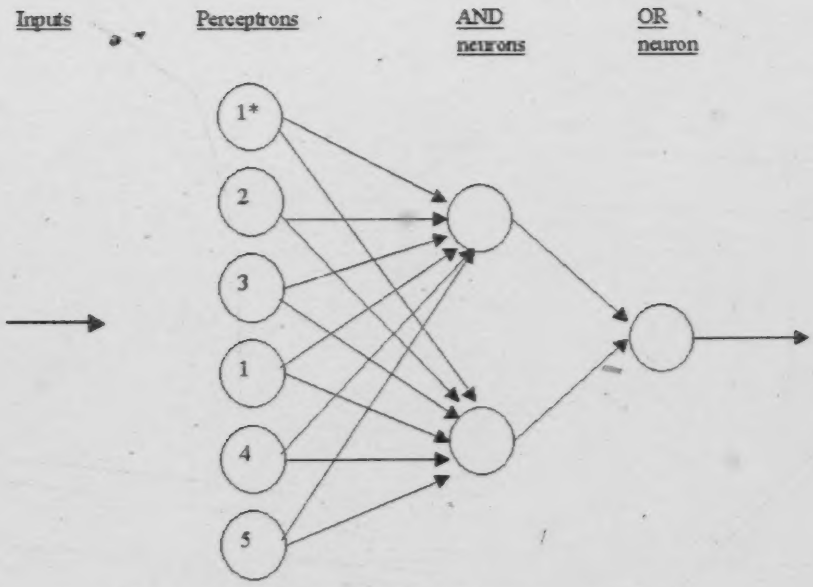


Figure 44: FALN equivalent to the linear classification tree

5.8 Experimental studies

This section presents a set of illustrative examples using two – dimensional synthetic data; the purpose is to visualize the most essential feature of the network and emphasize the logic processing aspects along with the interpretation side of the network.

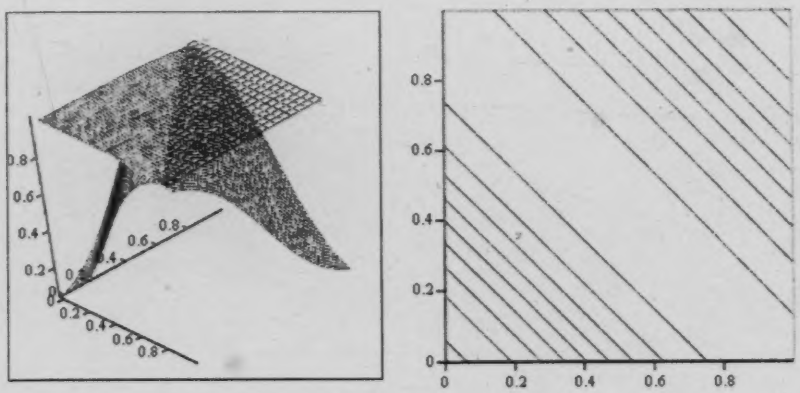
Experiment 1. Consider a simple two-dimensional Exclusive-OR (XOR) problem. The training is completed for $\alpha = 0.78$ and the size of the hidden layer and the number of perceptrons is set up to 1 and 2, respectively. The results of learning after 500 learning epochs (Q = 0.005314) are the following

Perceptrons:

$$U = \begin{bmatrix} -5.53 & -5.45 & 8.40 \\ 6.44 & 6.52 & -2.68 \end{bmatrix}$$

Logic processors: $V = [0.00 \quad 0.00]$; $W = [0.98]$

(as there is only one neuron in the hidden layer, the value of this connection is not relevant and it is close to 1). The plots of the characteristics of the resulting network are shown in Figure 45; observe that there are two perceptrons forming two separate regions in the feature space and then combined and-wise, using an AND fuzzy neuron.



(a) (b)

Figure 45: 3D and contour plot of the fuzzy adaptive network

Experiment 2 The data set shown in Figure 46 suggests a certain box-like geometry.

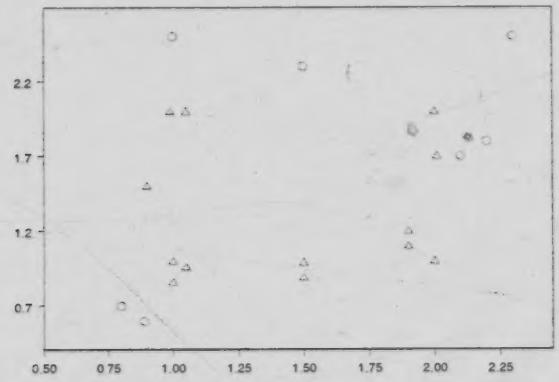


Figure 46: Two-dimensional two-class data (circles denote zeroes and triangles correspond to the target values equal to one)

In this case the structure of the network is experimented by changing the number of the perceptrons and varying the number of AND nodes in the hidden layer. The learning rate is 0.87 and the training was

carried out for 9,000 learning epochs. The resulting values of the performance index are summarized in the following table (here rows are indexed by the size of the hidden layer; hidden =1,2,…,7 while the columns correspond to the number of the perceptrons in the structure, p=1,2,…,10). The increase in the number of the processing nodes in general implies further reduction in the performance index. The minimal architecture that leads to low value of Q is selected as hidden=3 and p= 4.

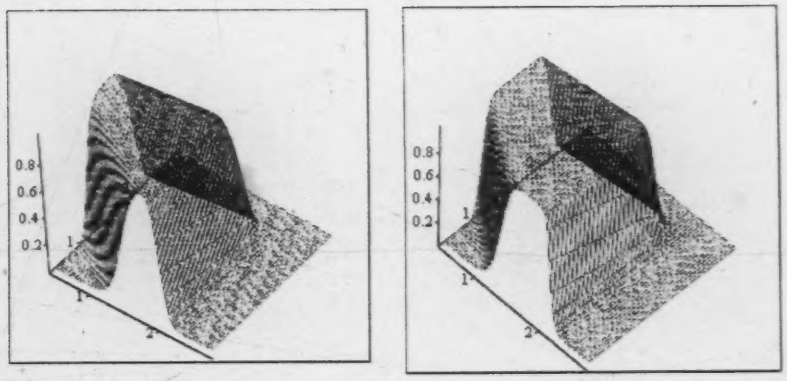
0.64	0.64	0.64	0.64	0.65	0.65	0.65	0.65	0.65	0.65
0.19	0.11	0.10	0.10	0.10	0.01	0.10	0.10	0.10	0.10
0.19	0.10	0.10	**0.01**	0.10	0.01	0.01	0.01	0.01	0.01
0.18	0.10	0.10	0.10	0.10	0.01	0.01	0.01	0.01	0.01
0.17	0.10	0.10	0.11	0.11	0.01	0.11	0.01	0.01	0.01
0.16	0.10	0.10	0.10	0.11	0.11	0.10	0.10	0.01	0.00
0.16	0.10	0.10	0.09	0.11	0.11	0.11	0.10	0.01	0.10

For this specific configuration of the network, the result is

$$U = \begin{bmatrix} -24.03 & -0.64 & 48.91 \\ 9.10 & 14.42 & -20.14 \\ 5.95 & 8.73 & -11.79 \\ 1.89 & -17.09 & 33.00 \end{bmatrix} \qquad V = \begin{bmatrix} 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 \end{bmatrix}$$

$$W = [0.95 \quad 0.95 \quad 0.95]$$

While there is a redundancy (as all AND neurons send the same signal to the OR neuron), the s-t composition realized by the OR neuron in the output layer gives rise to "sharper" edges of the construct. Figure 47 (a)-(b) contrasts these characteristics while the differences between the characteristics (absolute value of the difference between the output of the AND neuron and the entire network) are visualized in Figure 48.



(a) (b)

Figure 47: 3D characteristics of the AND neuron (a) and the network (b)

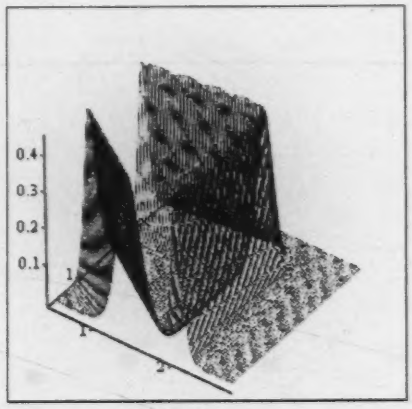


Figure 48: Difference between the characteristics of the AND neuron and the network

Experiment 3. This experiment is carried out using a subset of the spiral data presented in Chapter 3.

Obviously, with the increasing number of data (N), the geometric complexity of the problem increases. In the experiment, consider N =30 which leads to a subset of the already presented spiral, Figure 49.

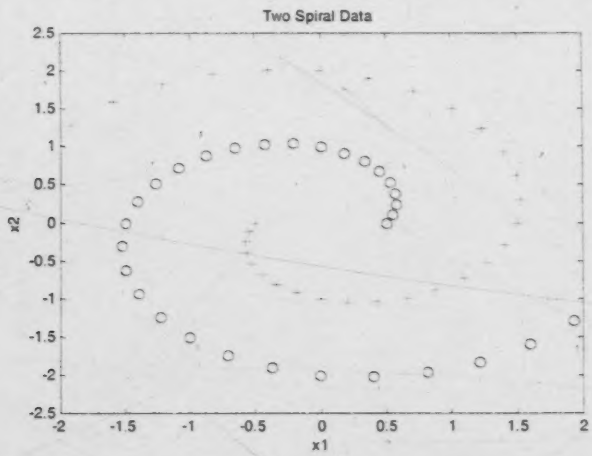


Figure 49: A two-class spiral data set

Figure 50 illustrates how the learning progresses (shown here are the values of RMSE in successive learning epochs while the initial configuration of the connections is random), for a network of 8 perceptrons and 3 AND-neurons (those are chosen arbitrarily but are quite close to the minimal required configuration required for solving this problem).

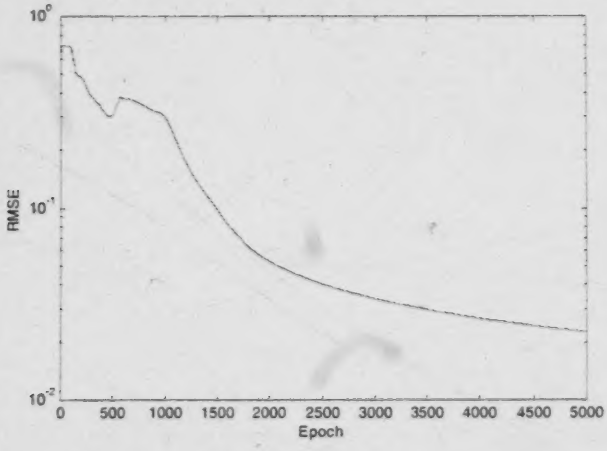


Figure 50: Performance index (RMSE) in successive learning epochs

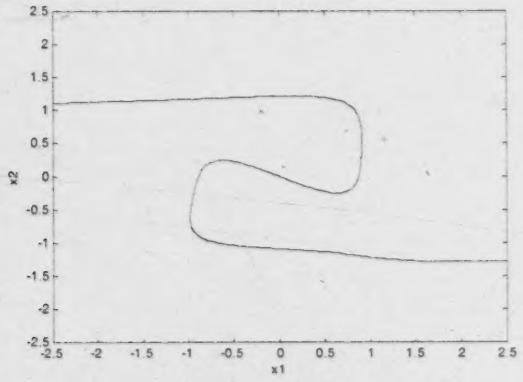
In the sequel, the arrays of the connections result as shown below

$$U = \begin{bmatrix} -3.02 & 4.32 & 0.53 & 0.06 & 3.36 & -0.26 & 4.77 & 4.19 \\ -5.14 & -0.52 & 8.56 & 3.59 & 3.74 & 5.88 & -0.12 & 2.31 \\ 0.01 & 4.00 & 9.34 & -0.68 & -1.28 & -7.13 & -4.30 & 0.02 \end{bmatrix}$$

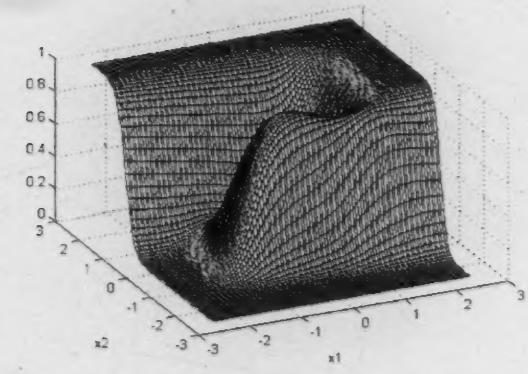
$$V = \begin{bmatrix} 0.00 & 0.00 & 0.00 & 1.00 & 1.00 & 1.00 & 1.00 & 1.00 \\ 1.00 & 1.00 & 0.00 & 1.00 & 1.00 & 1.00 & 0.00 & 1.00 \\ 1.00 & 1.00 & 0.11 & 0.60 & 1.00 & 0.00 & 1.00 & 1.00 \end{bmatrix}$$

$$W = [1.00 \quad 1.00 \quad 1.00]$$

The values of σ for the perceptrons are 2.75, 2.74, 9.03, 1.66, 2.34, 5.27, 3.02, and 2.60, respectively. Once learned, the decision surface produced by the FALN (as visualized in Figure 51 (a)-(b)) reflects the spiral nature of the data.



(a)



(b)

Figure 51: Plots of the decision boundaries formed by the FALN (contour and 3D plot)

Let us now consider the effect of parameter initialization on the performance of the network, namely the learning process itself and the obtained classification rate. Although random initialization may lead to successful training, it can also produce an initial weight configuration that is quite remote from the optimal solution. The case demonstrated here is the one of 7 perceptrons and 5 AND-neurons; as a matter of fact, a smaller network was found to be able to solve this problem. The weights of the OR neuron were fixed to be equal to 1. The case of unsuccessful training is illustrated in Figure 52 with three of the boundaries being far from the spiral. Successful training after 10,000 epochs carried out with a learning rate of 0.3 was achieved in about 20% of the experiments in which the weights were initialized randomly.

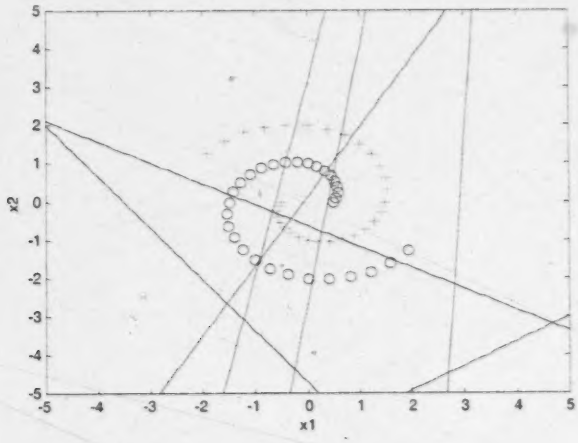


Figure 52: Example of random initialization of learning

The other initialization method is based on pseudo-inverse. If we assume, as a first approximation, that the problem is linearly separable (which is not the case here but can serve as a good approximation of the original problem), we complete the pseudo-inverse step on a randomly selected subset of 30 percent of the data set. The training was successful after 10,000 epochs, with a learning rate of 0.3 with positive results was achieved in about 80% of the experiments, see Figure 53.

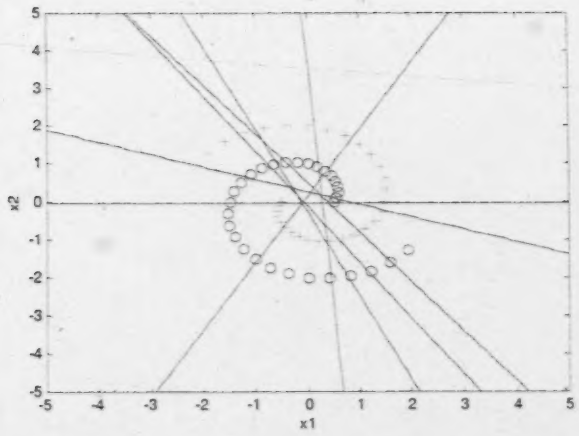


Figure 53: Example of pseudo-inverse initialization of learning

The structure of the network impacts its performance. So far it was shown that 7 perceptrons and 5 AND neurons led to successful learning. These experiments revealed that the increase of the size of the network did not adversely affect the classification of the classifier. To study the generalization capabilities of the FALN, the original set of patterns were modified by adding Gaussian noise with a standard deviation of 0.1. The results are visualized in Figure 54.

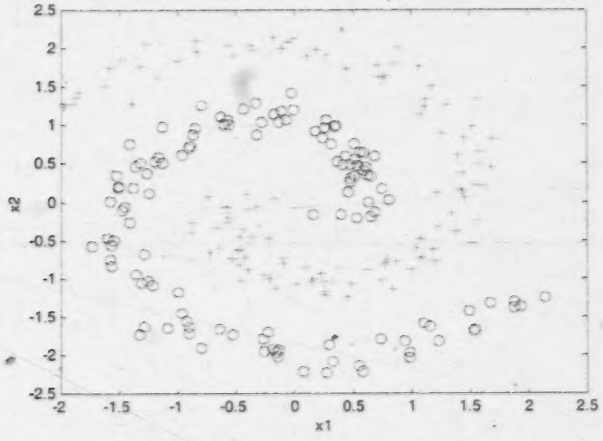


Figure 54: Noisy data set for evaluation of generalization capabilities of the network

To gain a comprehensive view at the performance of the network vis-à-vis selected parameters, the results

are collected in a tabular format, Table 14. All cases reported use a 5-fold cross validation technique with the results given in terms of the respective mean values and standard deviations (SD).

Type of learning: Incremental /Batch	Number of perceptrons.	Number of AND neurons	Learning rate	Training Epochs	Training: Classification rate	Testing: Classification rate
I	3	2	0.1	20,000	77.64±9.4	74.58±4.5
I	5	3	0.1	20,000	88.06±4.5	83.75±5.6
I	8	3	0.1	20,000	88.75±3.3	88.33±5.4
I	10	5	0.1	20,000	100.0	98.33±1.7
B	3	2	0.1	20,000	86.25±2.7	80.00±5.4
B	5	3	0.1	20,000	83.75±9.8	78.75±8.4
B	8	3	0.1	20,000	88.47±7.2	83.75±8.1
B	10	5	0.1	20,000	89.86±6.8	85.42±5.9

Table 14: Performance of FALN: (I- incremental learning; B- batch learning)

Experiment 4. The example discussed here is the two-dimensional linearly separable data, presented in Chapter 3, see also Figure 55. It is used as an example for illustrating the previously discussed three different initialization methods, and for showing the possible advantage in the MSM-T initialization scheme.

Consider this linearly non-separable classification problem. In order to design an FALN for solving this problem the number of perceptrons and neurons in each layer should be determined, and the weights should be initialized. Since this is a 2-dimensional piecewise linearly separable problem it may be assessed that 5 lines are sufficient for separating the 2 classes (of course this kind of observation would be impractical with high dimensional data).

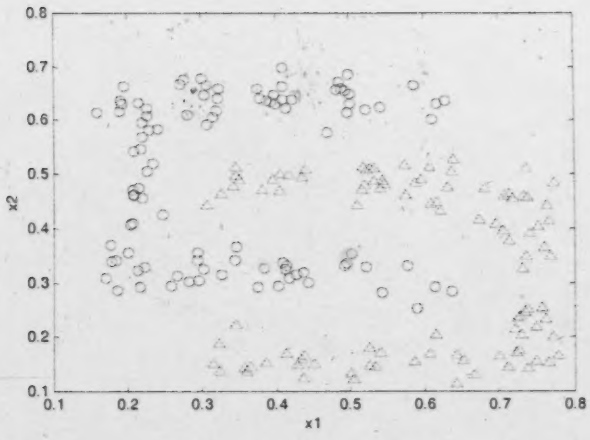


Figure 55: Piecewise linearly separable two-dimensional data

Random initialization. Using random initialization, even if the required number of lines is approximately correct, may lead to lines far from the classification boundaries. This leads to very inefficient learning and often convergence is not achieved at all.

Using the pseudo-inverse technique for initialization, the possible difficulty of random initialization (hyperplanes far from the classification boundary) is overcome easily. In order to avoid initializing all perceptrons with the same weights, each can be initialized with a solution of the pseudo-inverse to a subset of the data. Using a subset of the data can also be useful in case the data set is very large in which the calculation of the pseudo-inverse might cause numerical problems. However, the use of pseudo-inverse can help in initializing the weights at the perceptrons inputs better but does not help with the construction of the FALN. Figure 56 shows the lines generated using the pseudo-inverse method where for each pseudo-inverse calculation a subset of 10% of the data set was chosen randomly. Each of the lines crosses the classification boundary making the learning procedure to come, more efficient.

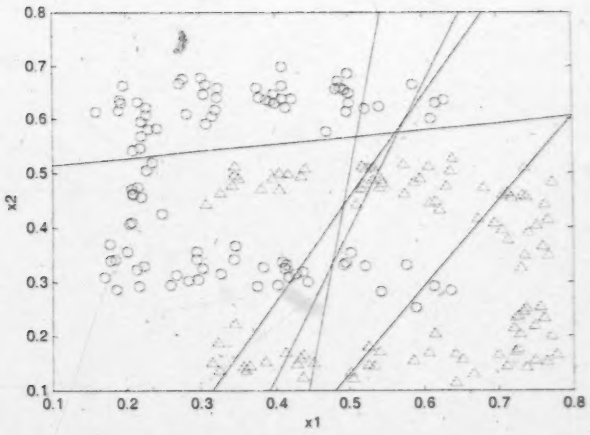


Figure 56: Piecewise linearly separable two-dimensional data; pseudo-inverse initialization

Initializing using the MSM-T algorithm (Bradley, 1995). The program was run on this data, setting the parameters to: maximum tree depth = 3; tolerance = 0.2. The resulting tree is of seven separating hyper-planes, shown in Figure 57, with a correct classification rate of 93%. Using the resulting classification tree to construct and initialize the FALN leads to efficient learning. It is worth noting at this point that the MSM-T program may also be run without imposing a limitation on the tree-depth and without allowing any tolerance, which will lead to perfect classification of the data set; however, the resulting tree might be unnecessarily too large, causing long training for the FALN and reduced generalization capability.

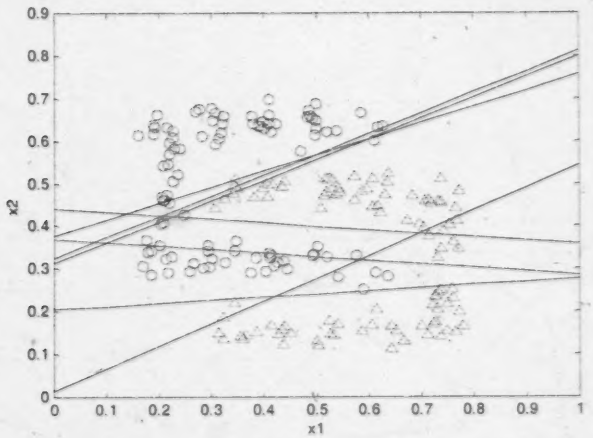


Figure 57: Piecewise linearly separable two-dimensional data; MSM-T initialization

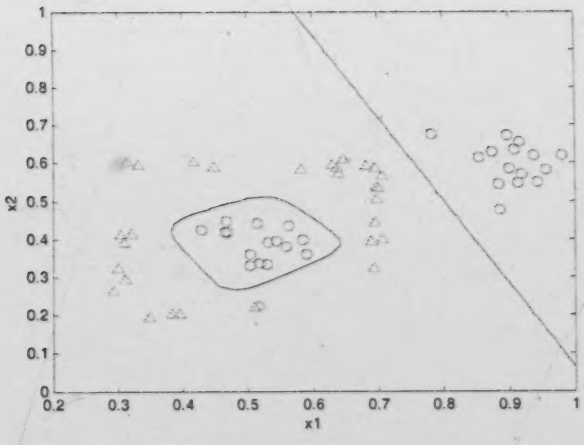
Table 15 demonstrates how learning efficiency can be affected by the initialization method. It summarizes

the results of training an FALN using the three discussed initialization techniques, in terms of the correct classification rate achieved for different values of training epochs (14 perceptrons; 4 AND neurons; learning rate 0.05). The results are averages over five experiments.

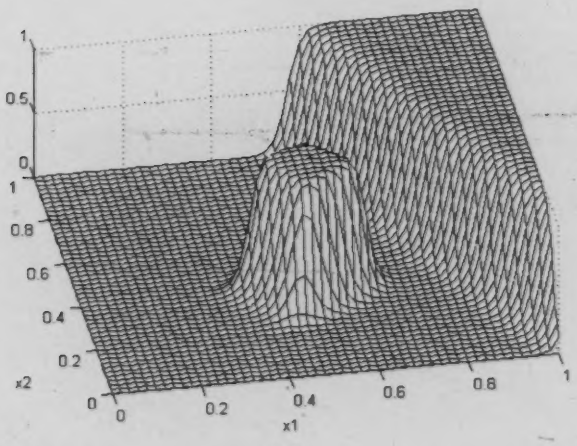
No. of epochs	Initialization method		
	Random	Pseudo-inverse	MSM-T
100	0.77	0.80	0.92
200	0.85	0.82	0.95
500	0.88	0.85	0.95

Table 15: The effect of initialization on learning

Experiment 5. Similar experiments were carried out for the data presented in the Figure 58 (a), which shows also the classification boundaries (Figure 57 (a)-(b)) that were achieved using the FALN. Initialization was done with the pseudo-inverse method and the network (8 perceptrons and 4 AND neurons) was trained for 20,000 epochs with a learning rate of 0.3.



(a)



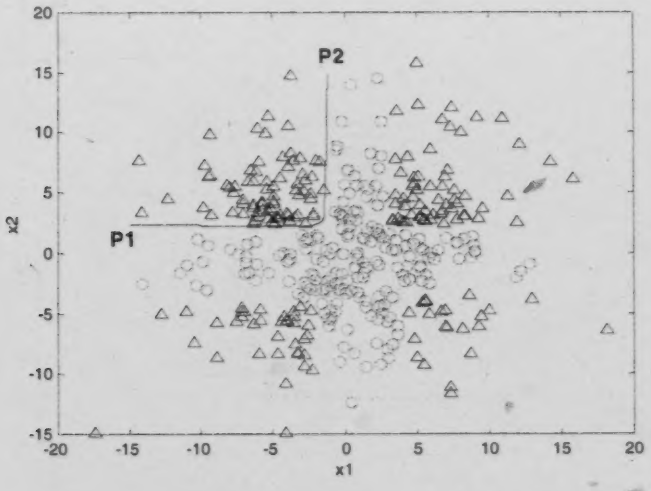
(b)

Figure 58: Experiment 4; synthetic data – two-class problem (a) and classification boundaries (b)

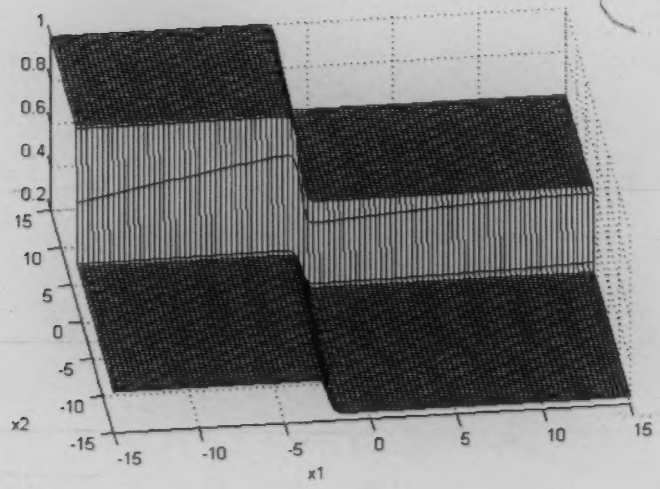
Experiment 6. Non-linear structured 2-dimensional artificial data

As an illustrative example, consider two-class FALNs for two-dimensional data. The intent is to visualize the geometry of the classifier and the role of individual perceptrons and AND neurons. The training was carried out for a few thousand epochs with the learning rate equal to 0.1. The results are shown in Figures 59-62. Starting with the simplest topology, FALN(2, 1) with only two perceptrons and a single AND

neuron, it can be seen that the underlying geometry of this classifier does not match the geometry of the data. This mismatch is manifested in the classification rate being equal to 0.76; as shown in Figure 59, there is a significant amount of misclassified points.



(a)



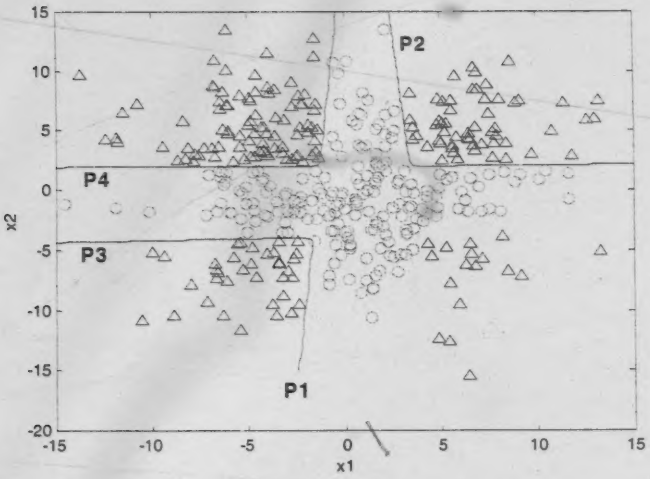
(b)

Figure 59: Two-class data and the performance of the FALN (2,1): (a) classification decision and (b) 3D plot of the classification characteristics

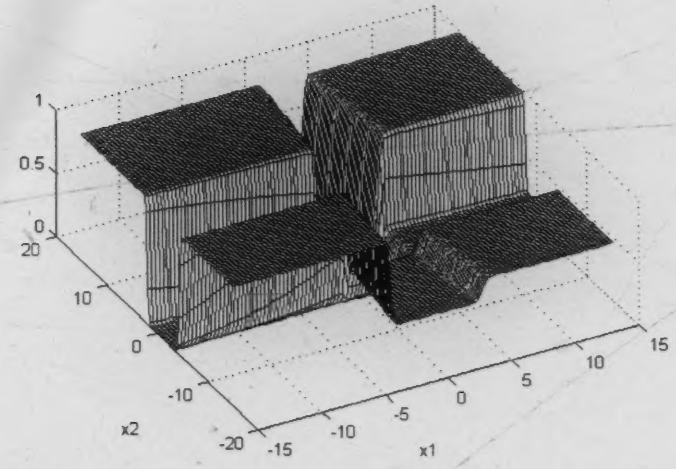
The resulting parameters of the perceptrons and AND neurons along with the values of σ are as follows

$$U = \begin{bmatrix} 0.01 & -1.18 \\ 1.48 & 0.03 \\ -3.44 & -2.38 \end{bmatrix}, V = [0.39 \quad 0.60], W = [1], \sigma = [10.92 \quad 6.02]$$

With the increase of the size of the network, it starts producing better results. It is shown in Figure 60 for FALN(4,3) where the classification rate is equal to 0.96. Subsequently, the FALN (5,3) shown in Figure 61 produces a classification rate of 0.99.

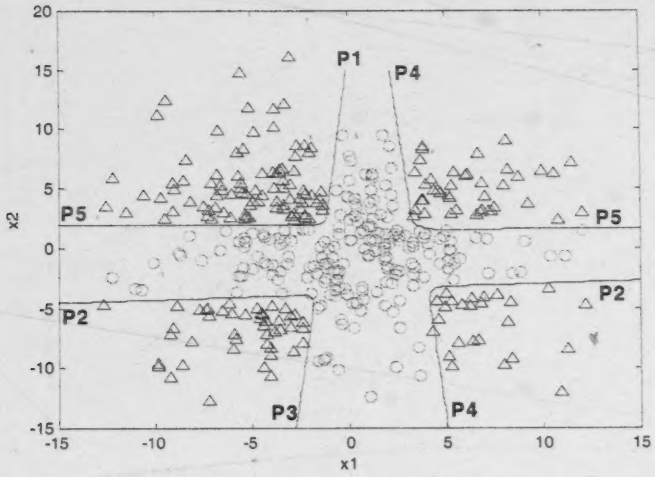


(a)

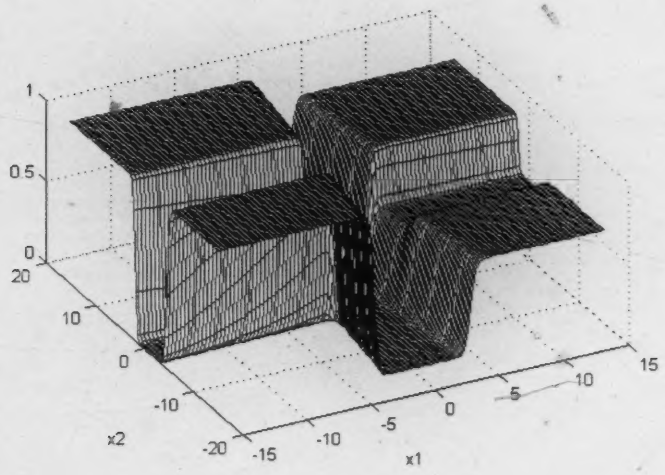


(b)

Figure 60: Classification with FALN(4,3): classification boundaries (a) and 3D characteristics of the classifier (b)



(a)



(b)

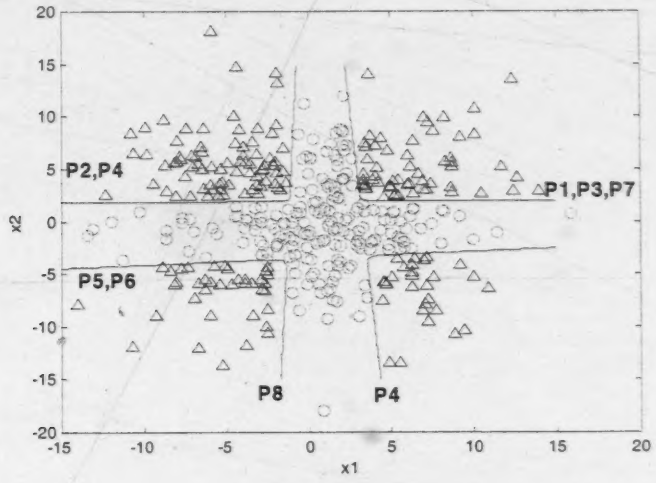
Figure 61: Experiments with FALN(5,3) with their decision boundaries (a) and 3D characteristics (b)

For FALN(5,3) the corresponding connections are given as follows

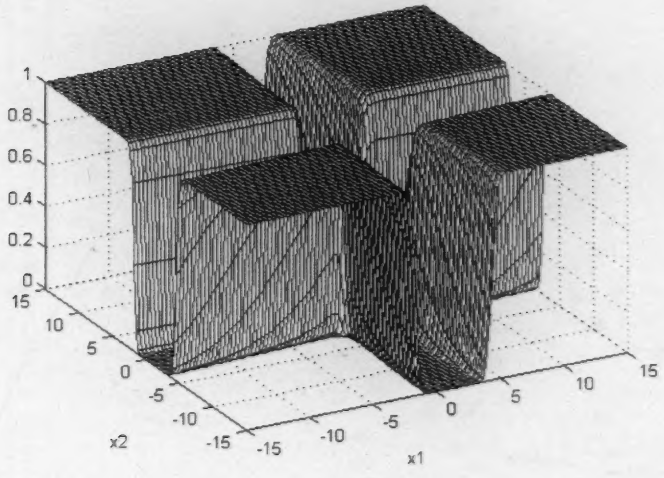
$$U = \begin{bmatrix} -0.92 & 0.02 & -0.93 & 0.53 & 0.00 \\ 0.07 & -0.46 & 0.09 & 0.05 & 0.81 \\ -1.06 & -1.76 & -1.21 & -1.80 & -1.61 \end{bmatrix}, V = \begin{bmatrix} 0.00 & 0.00 & 0.00 & 1.00 & 1.00 \\ 1.00 & 0.83 & 1.00 & 0.00 & 0.58 \\ 0.00 & 1.00 & 0.00 & 1.00 & 0.00 \end{bmatrix}, W = [1 \quad 1 \quad 1]$$

$$\sigma = [5.95 \quad 9.30 \quad 6.82 \quad 9.78 \quad 9.06]$$

The excessively large network such as e.g., FALN(8,4) produces "sharper" decision boundaries, however the result shows an obvious redundancy as presented in Figure 62, and that some other more compact networks already produced similar results.



(a)



(b)

Figure 62: FALN(8,4) with their decision boundaries (a) and 3D characteristics (b)

Experiment 7. Highly dimensional biomedical data

In this section, the experiments were carried out with patterns obtained from three complex biomedical data sets. They are interesting in several ways, especially by being highly dimensional and originating from real-world biomedical applications.

The first data set comprises 186 infrared (IR) spectra of synovial joint fluid, presented earlier in Chapter 3. The spectra are divided into one of three classes: 72 samples of rheumatoid arthritis, 72 samples of osteo-arthritis, and 42 control samples, Each The IR spectra cover the range 1000–3700 cm^{-1} and were discretized to 2801 data points. Because of the highly dimensional nature of these spectra, prior to any classification procedure, dimensionality reduction of the feature space was performed. As the spectra are quite smooth, a simple averaging technique was applied by averaging over a certain window of the spectral coefficients.

Determining of the window length "N" for averaging is done in a way that would lead to significant reduction of the feature space while keeping the reduced-dimensionality data set consistent with the original data set, in terms of maintaining the Euclidian distance structure within the data set. Therefore, in addition to the original highly dimensional data set, averaging N successive coefficients produces a second data set. The measure of consistency between the 2 data sets is computed as follows.

For each sample:

- In the original data set calculate the Euclidian distance to all other samples.
- In the reduced-dimensionality data set calculate the Euclidian distance to all other samples.
- Find what is the overlap (how many samples appear in both sets divided by the number of samples in a set) between the 2 following sets:
    1. The 5% of samples, which are nearest to the reference sample in the original data set
    2. The 5% of samples, which are nearest to the reference sample in the second data set
- Save the overlap measurement.
- Repeat the above operation for sets of 10%, 20%, 30%, 40% and 50% of the nearest neighbors to the reference sample.
- Find the minimum of the 6 resulting overlap values, and save it.

After completing this procedure, the average overlap value over all samples is calculated. The result is the measure of the consistency of the reduced-dimensionality data set with the original one, in terms of how well the distance matrix between samples is preserved.

Figure 63 displays graphically the results for the consistency measure for a range of window lengths.

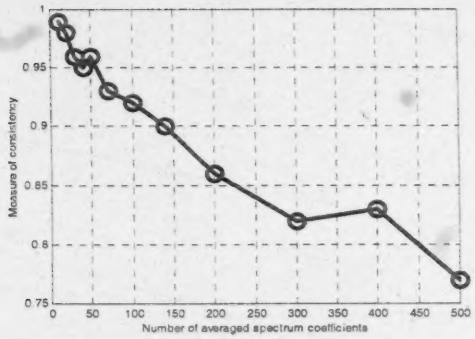


Figure 63: Consistency measure with respect to window length for averaging;
IR spectra of synovial fluid

With N = 70 the consistency measure is still quite high (0.93) while for N greater than 140 its value drops below 0.9. Thus the resulting data consists of samples containing 40 features.

The second data set comprises 206 magnetic resonance (MR) spectra of tissue biopsies from brain neoplasms, discussed in Chapter 3. The spectra fall into one of three groups: 95 spectra that correspond to meningiomas, 74 astrocytomas samples, and 37 spectra corresponding to non-tumourous brain tissue that served as control spectra. The spectra were acquired in the range 0.3–4.0 ppm, discretized to 550 data points and area-normalized.

Again, the principle of maintaining the measure of consistency was used a guideline for establishing the window length for averaging, as is shown in Figure 64 and it was chosen to be 5 resulting in data samples of 110 features.

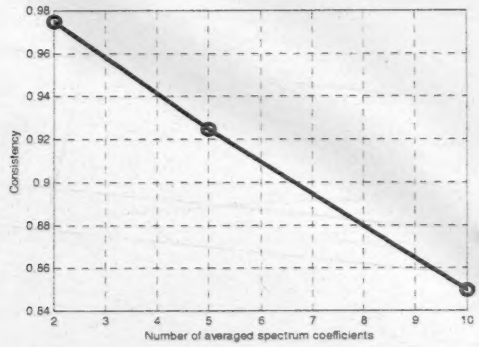


Figure 64: Consistency measure with respect to window length for averaging;
MR spectra of brain neoplasms

The third data set comprises 227 MR spectra of a biological fluid discretized to 512 points (also described previously in Chapter 3). The spectra were divided into one of three classes: 108 samples belonging to the class of normal patterns, 54 being of borderline character, and 65 abnormal patterns.

As this data is not nearly as smooth as the previous two data sets, no averaging of coefficient was performed; the data set was used as is in the classification process.

The performance of the FALN is compared with that of two well-known and extensively exercised classifiers: k-nearest neighbor (K-NN) (Cover and Hart, 1967; Dasarathy, 1991; Duda *et al.*, 2001), and linear discriminant analysis (LDA) (Duda *et al.*, 2001; Fisher, 1936; Fukunaga, 1990).

K-nearest neighbor

The k-nearest neighbor classifier is based on a simple algorithm that uses the training set, to classify samples from the testing, based on a distance measure. For each feature vector that it to be classified its Euclidian distance to all instances from the training set is computed, as follows where **x** and **y** represent two feature vectors consisting of "n" elements each.

$$d = \left[\sum_{i=1}^{n} (x_i - y_i)^2 \right]^{\frac{1}{2}}, \; x_i, \; y_i \in \mathcal{R}, \; i=1..n \tag{32}$$

Then, the "k" training set examples closest to it are found and the sample is assigned a class label according to the class that appears the most in these k training set instances.

LDA

There exist different formulations of LDA. The aim of LDA is to produce a linear transformation, denoted by a matrix of weights W that achieves maximal discrimination between data belonging to different classes, where W is an n-by-(k-1) matrix ("n" is the feature vector dimension, and "k" is the number of classes). In the 2-class case W reduces to a column vector **w**. LDA characterizes each class by its mean $\mathbf{m}_i$ (n-dimensional mean) and its covariance matrix S_i. Assuming the p_i is the a-priori probability of a sample belonging to class i, two scatter matrices are defined, the between-class scatter matrix S_B, and the within-class scatter matrix S_W. Focusing on the 2-class case we have (D_1 and D_2 correspond to data of class 1 and 2)

$$S_B = (\mathbf{m}_1 - \mathbf{m}_2)(\mathbf{m}_1 - \mathbf{m}_2)^T$$

(33)

and the scatter matrices S_i, S_w

$$S_i = \sum_{x \in D_i} (x - m_i)(x - m_i)^T$$

(34)

$$S_w = S_1 + S_2$$

(35)

where

$$m_i = \frac{1}{n_i} \sum_{x \in D_i} x$$

(36)

and the idea at the basis of this method is to find W that maximizes the ratio J(W), of the transformed between-class scatter to the transformed within-class scatter

$$J(W) = \frac{W^T S_B W}{W^T S_w W}$$

(37)

It can be shown that this leads to the criterion of assigning **x** to the class for which

$$L_i(x) = x^T S_w^{-1} m_i - \frac{1}{2} m_i^T S_w^{-1} m_i$$

(38)

is maximal.

The LDA is optimal under certain conditions being, the observations are of multi-variate Gaussian distribution, the covariance matrices for all classes are equal, and the different classes are of equal prior probabilities.

There exists an extension for the case of non-equal a-priori class probabilities

$$L_i(x) = x^T S_w^{-1} m_i - \frac{1}{2} m_i^T S_w^{-1} m_i + \log(p_i)$$

(39)

where "p_i" is the a-priori probability of a sample belonging to class "i".

Evaluation through cross-validation

In all cases, the experimental setup involves a 5-fold cross-validation. The patterns are split randomly into 5 subsets with 60% training, 20% validation and 20% testing set. The comprehensive results are collected in the series of tables shown below, Tab. 16-21. The following notation is used to denote different structures of the networks and various learning scenarios, that is

FALN(a, b) specifies the number of perceptrons (a) and AND neurons (b) used in the network. For instance, FALN(3, 4) means that the network has 3 perceptrons and 4 AND neurons. The symbols I and B refer to the mode of learning, that is Incremental (on-line) and Batch (off-line), respectively. To denote a way in which the learning process was initialized (that is randomly or through a pseudo-inverse) R and PI are used, respectively. The k-nearest neighbor classifier is parameterized by the number of neighbors, to emphasize the fact the notation K-NN(a) is used, with "a" being the number of nearest neighbors.

The detailed results are given for various network architectures. There is the issue of pre-determining the required network architecture for each specific problem, and that is where the MSM-T initialization method can offer its advantage. For each of the classification problems in these experiments the network structure resulting from the MSM-T algorithm is presented, however it is important to stress the point that other network configurations were experimented as well.

Classifier	Training set			Testing set		
	Class 1	Class 2	Class 3	Class 1	Class 2	Class 3
LDA	98.2±0.6	99.9±0.3	98.6±0.8	90.0±4.3	95.5±2.7	92.5±2.5
K-NN(3)	94.2±1.4	95.5±0.6	94.1±0.8	86.5±5.2	89.5±4.1	87.0±4.8
FALN (2,1)-I,R	94.5±2.4	92.6±2.1	86.0±1.5	86.0±6.5	83.0±9.2	85.5±13
FALN (2,2)-I,R	92.6±3.7	93.0±2.7	97.3±1.0	80.5±5.4	81.5±8.4	92.0±3.2
FALN (3,1)-I,R	92.1±2.7	92.6±2.4	93.2±3.1	81.5±5.8	83.5±8.4	87.5±7.3
FALN (2,1)-I,PI	98.1±0.7	99.6±0.5	98.9±0.4	90.0±5.3	91.0±7.8	91.5±2.8
FALN (2,2)-I,PI	99.2±0.8	100.0	99.6±0.8	90.0±6.4	90.5±6.0	91.0±2.2
FALN (3,1)-I,PI	98.7±0.8	99.8±0.4	99.1±0.7	88.5±6.5	90.0±7.0	91.0±1.4
FALN (3,2)-I,PI	99.2±0.8	100.0	99.8±0.4	90.5±5.7	90.5±6.0	91.0±5.2
FALN (3,3)-I,PI	98.7±0.8	100.0	99.2±0.4	90.5±4.8	92.0±6.9	92.0±1.1
FALN (2,1)-B,PI	98.3±0.8	99.2±1.0	98.9±0.8	90.0±5.3	91.5±6.7	90.5±2.7
FALN (2,2)-B,PI	99.2±0.8	100.0	99.6±0.8	90.5±4.4	90.5±5.9	91.0±2.2
FALN (3,1)-B,PI	98.9±0.8	99.8±0.4	99.4±0.5	89.5±6.5	90.5±6.0	91.0±1.4
FALN (3,2)-B,PI	99.2±0.8	100.0	99.8±0.4	90.5±6.5	90.5±7.4	90.0±4.3
FALN (3,3)-B,PI	99.2±0.8	100.0	99.8±0.4	89.5±6.5	90.5±7.0	90.5±4.5

Table 16: Classification rates (± standard deviation): Infrared spectra of synovial joint fluid

Using the MSM-T algorithm the resulting network is FALN(1,1), meaning that the MSM-T achieves perfect linear separation on the training set, thus one perceptron is sufficient, and the FALN is determined.

Network architecture = FALN(1,1)						
Classifier	Training set			Testing set		
	Class 1	Class 2	Class 3	Class 1	Class 2	Class 3
FALN(1,1)	100.0	100.0	100.0	89.5±6.0	92.0±9.6	91.5±2.8

Table 17: Classification rates (± standard deviation): Infrared spectra of synovial joint fluid with FALN determined and initialized using MSM-T

Classifier	Training set			Testing set		
	Class 1	Class 2	Class 3	Class 1	Class 2	Class 3
LDA	100.0	99.4±0.6	98.6±0.5	89.3±4.2	73.9±6.2	66.5±4.5
K-NN(5)	88.5±2.4	86.3±3.4	86.1±1.2	81.9±6.2	82.8±7.6	78.6±5.3
FALN (2,1)-I,R	90.2±2.0	90.7±3.3	83.0±0.8	83.3±17	74.5±6.7	80.5±2.1
FALN (2,2)-I,R	92.2±11	89.2±5.0	84.3±2.1	84.2±1.5	76.3±6.9	80.5±1.3
FALN (2,1)-I,PI	100.0	99.5±0.8	95.7±1.4	94.9±2.5	82.3±4.8	72.5±6.4
FALN (3,1)-I,PI	100.0	99.3±0.7	97.8±1.2	94.9±1.9	76.2±3.4	73.5±7.3
FALN (4,1)-I,PI	100.0	99.3±1.6	97.7±2.1	93.5±1.9	77.7±5.1	78.1±6.7
FALN (4,4)-I,PI	100.0	98.8±0.9	98.7±1.4	94.4±3.0	80.5±7.1	75.8±6.1
FALN (2,1)-B,PI	100.0	99.5±0.7	96.2±1.9	94.9±2.6	81.4±4.4	73.5±7.1
FALN (3,1)-B,PI	100.0	99.2±0.6	98.3±0.6	94.9±1.9	77.7±3.5	72.6±7.8
FALN (4,1)-B,PI	100.0	97.8±0.9	98.5±1.1	93.5±1.9	78.1±5.8	78.1±7.8
FALN (4,4)-B,PI	100.0	96.0±2.2	93.3±4.0	94.9±3.8	84.9±7.6	78.6±6.7

Table 18: Classification rates (± standard deviation): MR spectra of brain neoplasms

Using the MSM-T algorithm the resulting network is FALN(1,1), meaning that the MSM-T achieves perfect linear separation on the training set, thus one perceptron is sufficient, and the FALN is determined.

Network architecture = FALN(1,1)						
Classifier	Training set			Testing set		
	Class 1	Class 2	Class 3	Class 1	Class 2	Class 3
FALN(1,1)	100.0	100.0	100.0	95.3±3.3	78.6±5.8	72.6±8.1

Table 19: Classification rates (± standard deviation): MR spectra of brain neoplasms with FALN determined and initialized using MSM-T

Classifier	Training set			Testing set		
	Class 1	Class 2	Class 3	Class 1	Class 2	Class 3
LDA	98.7±0.2	98.5±0.3	96.5±0.6	70.4±4.2	59.2±7.0	55.4±4.4
K-NN(11)	71.1±2.7	76.8±1.5	73.7±1.5	61.1±8.5	75.8±2.2	70.0±3.5
FALN (2,1)-I,PI	99.8±0.4	100.0	99.8±0.4	74.2±10	67.3±5.6	60.8±4.6
FALN (3,1)-I,PI	100.0	99.8±0.4	98.5±1.4	75.4±7.7	65.8±6.3	58.9±7.3
FALN (2,2)-I,PI	99.7±0.7	99.7±0.4	99.2±0.6	73.8±10	68.5±4.2	60.8±5.2
FALN (3,3)-I,PI	100.0	98.9±1.8	97.9±1.7	76.9±9.9	71.2±6.3	58.5±7.8
FALN (4,4)-I,PI	100.0	99.2±1.8	97.6±2.1	76.6±8.5	70.4±5.4	63.1±5.3
FALN (2,1)-B,PI	99.0±0.7	91.2±7.0	92.8±1.4	74.2±7.8	71.5±6.4	61.2±6.8
FALN (3,1)-B,PI	98.2±0.9	98.4±1.7	93.8±4.1	76.6±5.7	67.3±6.5	58.5±15
FALN (2,2)-B,PI	95.8±1.8	93.3±1.8	90.7±2.5	77.7±6.0	70.0±3.5	63.9±12
FALN (3,3)-B,PI	98.7±1.4	97.9±2.2	96.7±0.8	76.2±7.3	72.7±6.4	61.2±5.0
FALN (4,4)-B,PI	98.4±1.4	96.4±3.5	96.7±2.1	76.9±11	71.5±3.7	62.3±2.9

Table 20: Classification rates (± standard deviation): MR spectra of a biological fluid

Using the MSM-T algorithm the resulting network is FALN(1,1), meaning that the MSM-T achieves perfect linear separation on the training set, thus one perceptron is sufficient, and the FALN is determined.

Network architecture = FALN(1,1)						
Classifier	Training set			Testing set		
	Class 1	Class 2	Class 3	Class 1	Class 2	Class 3
FALN(1,1)	100.0	100.0	100.0	72.7±9.2	61.9±4.6	52.3±6.4

Table 21: Classification rates (± standard deviation): MR spectra of a biological fluid with FALN determined and initialized using MSM-T

From Tables 16-21 it can be concluded that although the MSM-T algorithm provides a linear solution to each of the three classification problems, the optimal FALN (in terms of generalization capability) is more complex and is obtained empirically. A good example for this is the MR spectra of a biological fluid when comparing the performance of FALN(4,4) and the FALN obtained using MSM-T.

Experiment 8. Machine Learning data

For illustrative purposes, two data sets coming from the Machine Learning repository at UC at Irvine were experimented with, that is wine and Pima Indian diabetes (Murphy and Aha, 1994). In these experiments the pseudo-inverse was employed as an initialization method and various network architectures were evaluated. The results are summarized in Tables 22 and 23 below.

Classifier	Training set			Testing set		
	Class 1	Class 2	Class 3	Class 1	Class 2	Class 3
LDA	100.0	98.3±0.8	98.6±1.1	98.9±2.4	96.2±3.1	96.8±2.2
K-NN(3)	98.4±0.8	97.0±0.6	98.7±0.3	98.4±1.5	96.7±1.2	98.4±1.5
FALN (2,1)-I,PI	100.0	100.0	100.0	98.9±2.4	96.7±3.5	98.9±1.5
FALN (3,1)-I,PI	100.0	100.0	100.0	98.9±2.4	96.7±3.5	98.9±1.5
FALN (2,2)-I,PI	100.0	99.8±0.4	100.0	98.9±2.4	96.7±3.5	98.9±1.5
FALN (3,3)-I,PI	100.0	99.8±0.4	99.8±0.4	99.4±1.2	96.7±3.5	99.4±1.2
FALN (2,1)-B,PI	100.0	100.0	100.0	98.9±2.4	96.8±3.5	98.9±1.5
FALN (3,1)-B,PI	100.0	99.8±0.4	99.8±0.4	98.9±2.4	96.8±3.5	98.9±1.5
FALN (2,2)-B,PI	100.0	99.8±0.4	100.0	98.9±2.4	96.7±3.5	98.9±1.5
FALN (3,3)-B,PI	100.0	100.0	100.0	99.5±1.2	96.8±3.5	98.9±1.5

Table 22: Classification rates (± standard deviation): Wine data.

Classifier	Training set	Testing set
LDA	68.9±1.5	68.0±1.0
K-NN(9)	80.3±1.5	75.5±5.1
FALN (2,1)-I,PI	80.1±3.2	75.5±3.3
FALN (3,1)-I,PI	79.6±2.7	75.3±2.6
FALN (2,2)-I,PI	80.1±1.1	75.0±2.9
FALN (3,2)-I,PI	80.1±2.4	76.3±3.1
FALN (10,8)-I,PI	78.8±1.3	78.1±2.4
FALN (15,12)-I,PI	79.6±1.9	78.1±2.3
FALN (2,1)-B,PI	80.1±2.3	75.0±3.8
FALN (3,1)-B,PI	79.6±2.1	74.8±3.7
FALN (2,2)-B,PI	78.5±2.4	74.6±3.7
FALN (3,2)-B,PI	80.4±1.0	75.1±2.4
FALN (10,8)-B,PI	78.9±1.2	78.5±1.9
FALN (15,12)-B,PI	79.4±2.0	78.9±2.2

Table 23: Classification rates (± standard deviation): Pima Indian diabetes data

The experiments lead to a number of interesting findings. First, that FALNs are highly non-linear classifier (as exemplified by several two-dimensional problems) and this contributes to the high flexibility of such architectures. The second concerns the impact sound initialization of the parameters of the classifiers has on its performance. Thirdly, a superior performance of the FALNs with respect to some other types of classifiers is observed.

5.9 Summary

This chapter included a thorough study of the FALN. Following a discussion of the theoretical aspects, a wide-ranging series of experiments was exhibited. These experiments demonstrated the FALN's ability to learn, as well as its effectiveness as a highly non-linear classifier. Through a wide selection of real-world data it was shown that its performance is often superior with respect to other well-known classifiers.

6. Ensembles of FALNs

6.1 Introduction

This chapter describes a first attempt to combine a collection of FALNs in an ensemble. The chapter offers first a review of a family of techniques known as *ensemble methods* and their guiding principles are highlighted. In the second part of this chapter a series of experiments involving ensembles of FALNs are described.

6.2 Ensemble methods

The term *ensemble methods* refers to the techniques of combining the results of multiple classifiers to produce a single classifier. A general architecture of the ensemble is shown in Figure 65. It has been justified theoretically and established in many experimental studies (Bauer and Kohavi, 1999; Dietterich, 2000; Hansen and Salamon, 1990; Maclin and Opitz, 1997; Opitz and Maclin, 1999; Quinlan, 1996) that an ensemble of classifiers produces more accurate results than any of the individual classifiers making that ensemble. The two most popular methods of developing accurate classifier are *bagging* (Breiman, 1996a) and *boosting* (Freund and Schapire, 1996). Bagging is a method that relies on re-sampling the original training and thus building different training sets for each of the classifiers. Boosting methods can be divided into two main types: one of them is also based on re-sampling, though conceptually different than bagging, while the other is based on weighting the instances of the training set. There has been a substantial body of theoretical and experimental evidence in support of the effectiveness of the ensemble methods. One such comprehensive study (Opitz and Maclin, 1999) describes a thorough evaluation of both bagging and boosting on 23 data sets coming from the UCI Machine Learning repository (Murphy and Aha, 1994). Experiments were performed using these ensemble methods in the context of neural networks and decision trees (being viewed as two popular classes of classifiers).

Being more specific, the ensemble methods are divided into two types:

- individual classifiers are constructed on the basis of different randomly drawn subsets of the training data — bagging
- importance (relevance) of individual instances of the training is set up adaptively, according to the performance of the previously designed classifiers — boosting

The results of these studies (Bauer and Kohavi, 1999; Dietterich, 2000; Hansen and Salamon, 1990; Maclin

and Opitz, 1997; Opitz and Maclin, 1999; Quinlan, 1996) help establish several general conclusions. First, bagging generally produces a better classifier than any of the contributing classifiers. For boosting, however, the results could vary. For a few data sets, boosting improved performance of the resulting (sometimes being better than the result obtained for bagging), but for others it actually decreased the classification rate. Additionally, as evidence indicates (Dietterich, 2000; Maclin and Opitz, 1997; Opitz and Maclin, 1999), boosting may be sensitive to noise, which may, as a result, lead to increased error rates. As to the question of how many component classifiers should be used in an ensemble, different studies (Freund and Schapire, 1996; Opitz and Maclin, 1999; Quinlan, 1996) indicate that considerable improvement can be seen up to a total of approximately 25 classifiers (again these results were obtained using neural networks or decision trees). Obviously, these findings could be different when dealing with other categories of classifiers. It is worth stressing that the general model outlined in Figure 64 deals with any type of the classifier; in general the classifiers are considered to be homogeneous (for instance, all are neural networks). Referring to the same figure, a typical way of combining the classifier is a standard averaging. It becomes obvious that combining several identical classifiers (in their results) does not lead to any advantageous effect. It has been shown empirically (Opitz and Maclin, 1999) that using ensembles results in improvement when the individual classifiers disagree between themselves.

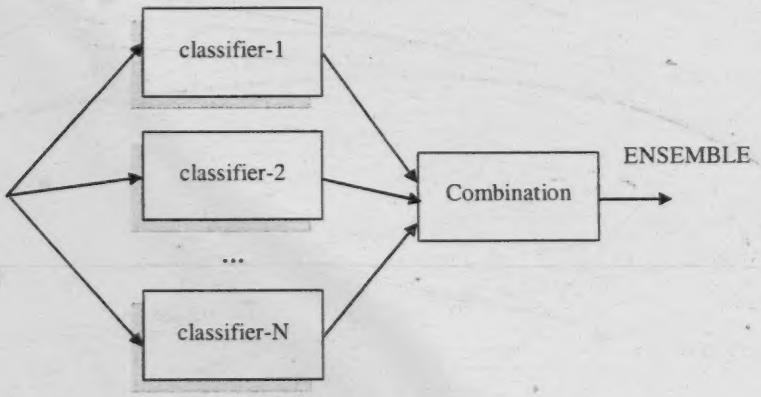


Figure 65: A classifier ensemble of neural networks

In what follows, the algorithmic issues of bagging and boosting are discussed in order to shed light on the essence of classifier combination.

6.2.1 Bagging

The term bagging comes from *bootstrap aggregation*. Bagging (Breiman, 1996a) is an ensemble method

that creates different component classifiers by training each classifier on different bootstrap samples of the training set. Each bootstrap sample is created by randomly drawing, with replacement, N patterns, where "N" is the size of the original training set. The resulting training set may include duplicate patterns from the original training set of patterns, while others may be excluded. For a given bootstrap sample, a pattern in the training set has probability $1-(1-1/N)^N$ of being selected at least once within the N random selections. For large N, this is about 63.2% (that is, each bootstrap sample contains about 63.2% unique instances from the training set. This perturbation causes different classifiers to be built if the training algorithm is unstable (meaning that modified training data sets lead to a variety of classifiers; e.g. neural networks or decision trees) and the performance can improve if the produced classifiers are good and uncorrelated.
The bagging algorithm in a pseudo-code form is provided in Table 24 below.

Let T be the number of classifiers in the ensemble

For $i = 1,\ldots,T$

- Create bootstrap sample i from the original training set (sample with replacement)
- Train classifier i using this bootstrap sample

Decide on the classification result according to a majority vote

---

Table 24: The Bagging algorithm pseudo-code

6.2.2 Boosting

Boosting (Freund and Schapire, 1996; Schapire, 1990) refers to a set of methods, and is reviewed extensively by Schapire (2002). The basic idea of these methods is to produce classifiers in series, rather than in parallel as done in bagging. The training set used for each member of the series is chosen based on the performance of earlier classifiers in the series. In boosting, patterns that are incorrectly predicted by previous classifiers in the series are either chosen more often than patterns that were correctly predicted, or given higher importance through a weighting mechanism. Denoting the classifier index with "t", boosting designs the training set for classifier t+1 based on the performance of the current t classifiers. Thus boosting, at each stage, tries to produce a new classifier that is a superior predictor for those patterns for which the current ensemble's performance is low, while in bagging re-sampling is independent on the performance of other classifiers in the ensemble.
The main boosting algorithms are AdaBoost (Freund and Schapire, 1996) and *arcing* (Adaptively resample and combine) (Breiman, 1996b). AdaBoost can use two approaches: (a) selecting a set of examples based on the probabilities of the patterns, or (b) using all of the patterns and weight the error of each pattern by its

probability (that is, patterns with higher probabilities have more effect on the error). The second approach has the advantage that all patterns participate in the training. Similar to bagging, arcing involves selecting a training set of size N for classifier t+1 by probabilistically selecting (with replacement) patterns from the original N training patterns. Unlike bagging, however, the probability of selecting a pattern is not equal across the training set. This probability depends on how often the pattern was misclassified by the previous t classifiers.

The pseudo-code for the AdaBoost algorithm (first approach) is given in Table 25 below.

Given pairs of patterns and bi-polar class labels

$(\mathbf{x}_1,y_1), \ldots (\mathbf{x}_N,y_N)$ where $\mathbf{x}_i \in \mathcal{R}^n$, $y_i \in \{-1,+1\}$, i=1..N

Initialize a probability function D_1 defined over the training patterns by assigning the same weight to each pattern, that is $D_1(i) = 1/N$

For t = 1, …, T *// main iteration loop*

- Train classifier h_t, using distribution D_t
- Assess the performance of the classifier on each pattern. The classifier produces the input-output relation: $h_t : \mathbf{x} \rightarrow y$ whose classification error is expressed in the form $\varepsilon_t = \Pr [h_t (\mathbf{x}_i) \neq y_i]$ with the calculations taken over the training set. Compute the expression

  $$\alpha_t = \frac{1}{2} \ln(\frac{1-\varepsilon_t}{\varepsilon_t})$$

- Update the probability function (weighting scheme applied to each pattern in the training set)

  $$D_{t+1}(i) = D_t (i) / Z_t \exp[\alpha_t y_i h_t (\mathbf{x}_i)]$$

  where Z_t is a normalization factor (that is selected in such a way that the resulting D_{t+1} becomes a probability function).

The aggregation of the classifiers is completed using a weighted voting scheme, where the weight of the individual classifier h_t is taken as α_t.

Table 25: Pseudo-code for the AdaBoost algorithm

AdaBoost calls for the given learning algorithm to be repeated in a series of rounds (by varying the values of "t" from 1 to T). One of the main ideas of the algorithm is to develop a distribution of weights over the training set where a weight value attached to a certain pattern (data point) denotes its relevance in the learning process. The distribution weight of a training pattern, in round t is denoted by $D_t(i)$. Initially, all weights are equal (so all patterns are treated in a uniform manner). In successive rounds, the weights of incorrectly classified patterns are increased so that the base learner (viz. the learning algorithm) is forced to focus more actively on the patterns that are more difficult to classify. The base learner's task is to produce a classifier appropriate for the distribution D_t, i.e. a classifier that will minimize the classification error on the

data set derived using D_t. Once the classifier h_t has been developed, AdaBoost chooses a certain parameter α_t which (heuristically) measures the importance that it assigns to h_t. For the 2-class problem it is typically set to

$$\alpha_t = \frac{1}{2}\ln(\frac{1-\varepsilon_t}{\varepsilon_t}) \tag{40}$$

(there exists an extension of this method to the multi-class case). The distribution D_t is then updated. The final classifier produces a weighted majority vote of the basis of "T" classifiers where α_t is the weight assigned to h_t.

The second version of the AdaBoost algorithm (AdaBoost.M1), (Freund and Schapire, 1996) also changes the weights of the training instances, provided as input to each training algorithm based on classifiers that were previously built.

6.3 Experimental studies

6.3.1 Medical Imaging System (MIS) data

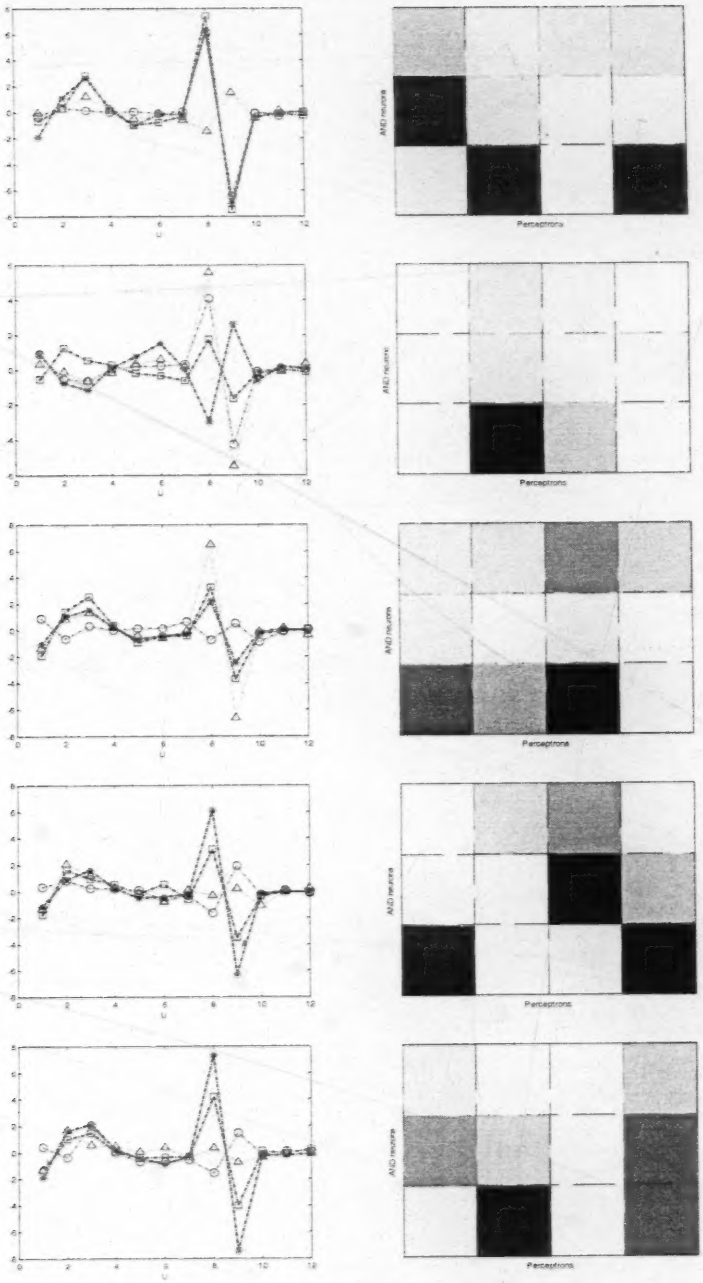
It goes without saying that software quality becomes an ultimate goal of all software endeavors. It becomes of paramount importance to be in a position to assess such quality in a quantitative fashion. One aspect of this is the growing interest in the capability of assessing and predicting a software project's reliability, which is strongly associated with the effort that will be required to make it operative. Consequently, there is much importance in researching methods for predicting quantitatively, how many faults a software module is likely to have. To address this issue, we need to be able to build models of software quality that form a sound framework within which one can make reliable predictions as to the quality of software products or processes.

The goal is to develop a prediction model of software quality in which the number of modifications (changes) is projected on a basis of the values of the 11 software metrics that is used to characterize a software module. The problem is formulated in the setting of classification and because of this assumed format, the output variable (number of changes) is discretized (categorized) by distinguishing between several limited, but intuitively appealing, categories (say, class_1: software modules with no changes; these could be eventually deemed to be fault-free, class_2: software modules with number of modifications between 1 and 10, and software modules with the number of changes over 10; this category can be sought as potentially faulty modules where most of the testing and maintenance effort should be focused).

So far, the architecture of the network was discussed and a comprehensive design methodology was developed including the fundamental technique of bagging and boosting. All of these are now applied to the development of the software quality model implemented in the form of the FALN. The AdaBoost algorithm was considered as the implementation of the boosting approach. As indicated, the classification of the MIS data is discussed by looking at the discrimination between software modules on the basis of the values of the software measures. To increase reproducibility of the results, the experimental setup involves a 5-fold cross-validation. The comprehensive results are collected in the series of tables shown below, Tables 26–36 conform to the notation already used in the previous sections, that is in the expression FALN(a, b) the number of perceptrons is denoted by (a) and AND neurons by (b). Similarly, bagging(3, 4) denotes the use of the bagging method for the network with 3 perceptrons and 4 AND neurons. The same notation applies to the boosting technique. In the rows of these tables that concern the results of bagging and boosting, an additional number (in parenthesis) indicates the number of networks with which the ensemble achieved the highest accuracy. Fig. 70-73 display examples of error-rates achieved using bagging and boosting with respect to ensemble size, with the different MIS data classification problems. Fig. 75-77 display examples of error-rates similarly, with biomedical data classification problems.

To compare the FALN, another standard classifier is used, the K- nearest neighbor (K-NN), which has an extremely simple architecture. To emphasize the number of neighbors, the notation K-NN(a) is used, with "a" denoting the number of nearest neighbors. To initialize the weights of the perceptron, the standard pseudo-inverse is used. The learning is done in an incremental mode. As stated, the original MIS data set was treated in the setting of the classification problem by forming some categories of the fault-proneness of the software modules. The series of experiments completed here deals with several ways of categorizing the software modules.

As this is a new type of classifier, which has not been previously investigated, it is beneficial to gain a better insight into the nature of the bagging and boosting completed for FALNs. Figure 66 displays the weight connection parameters of the networks of the resulting ensemble formed through the bagging process. The values of the parameters of the perceptrons (denoted U) are shown in the left-hand side of the figure while the weight connection values of the AND neurons (symbolized by V) are visualized through a series of two-dimensional arrays (the level of brightness corresponds to the values of the connections; higher values are mapped on darker shades of the matrix). It is apparent that the ensemble has a substantial variability in the connections, especially those associated with the AND neurons.


AND neurons
Perceptrons

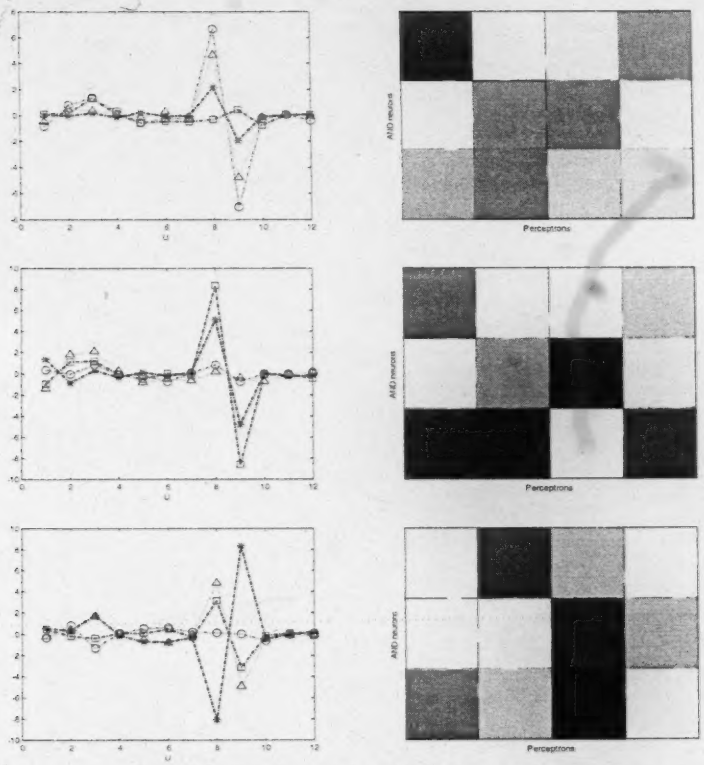


Figure 66: Values of connections (U - left and V - right) for eight different FALNs (represented in eight rows) being used in the ensemble formed by the bagging method

With the bagging method, there is significant variation in the resulting networks (which is somewhat random in its nature). Figure 67 shows how the weighting of the individual data changes from step to step. Interestingly, in contrast to the bagging strategy, the dispersion in the values of the connections (V) is considerably smaller.

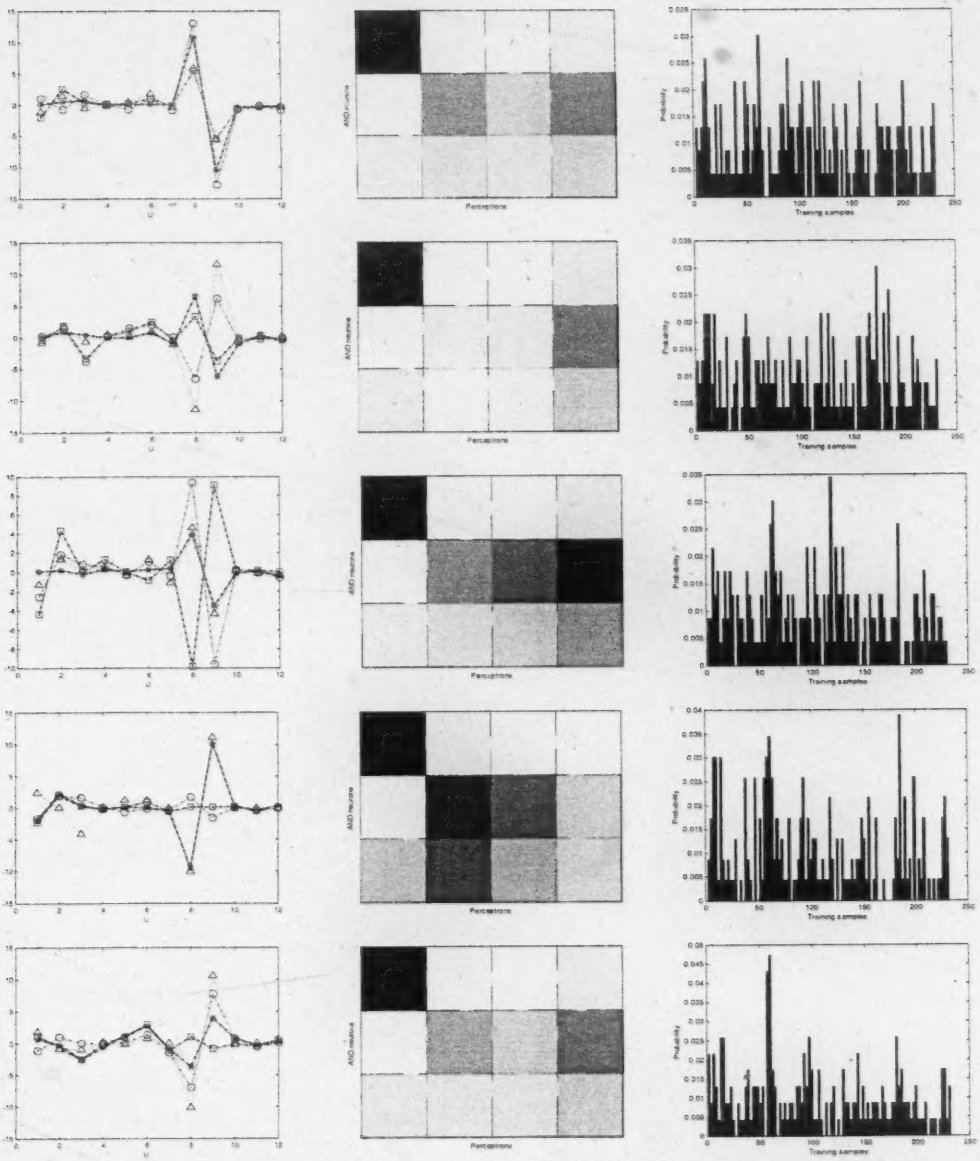


Figure 67: Boosting method: values of the connections of five FALNs (in successive rows) (U-left and V – middle) along with the distribution of the weights attached to the individual data points

Upon careful examination of the distribution of weights, elements in the data set that are the most difficult to classify can be identified. This helps to carry out their detailed analysis and in this way learn about their specificity and investigate their character as a potential "outliers" to be excluded from the analysis. As

revealed by Figure 67, one of the data points (numbered as 58) is an obvious "difficult" case to be classified. This data point is examined by looking at the values of the features and contrasting these with other patterns belonging to the first class, see Figure 68. It became apparent that this data point is not typical for the class to which it was assigned. The differences are also present when superimposing the same data point on the testing set with the results visualized in Figure 69. Furthermore, an analysis of the results on the testing set achieved in this experiment shows that after 5–8 iterations of the boosting method, the performance of the classifier starts to deteriorate. In order to understand this refer to Figure 69, which displays the data point under discussion together with the 23 other examples coming from the testing set and labeled to belong to class 1. Again, it can be observed that the data point is quite atypical in comparison to the rest of data. It is worth noting that for such data points, boosting may exhibit some deterioration in performance over successive iterations.

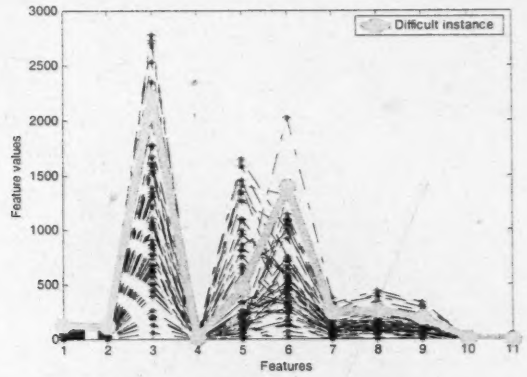


Figure 68: "Difficult" (to classify) data point in the feature space; to facilitate its contrasting with other data points, their feature values are also included in the plot.

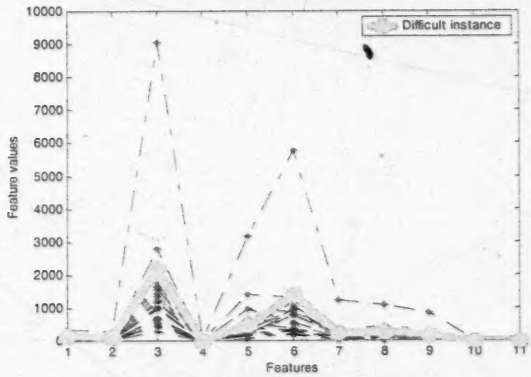


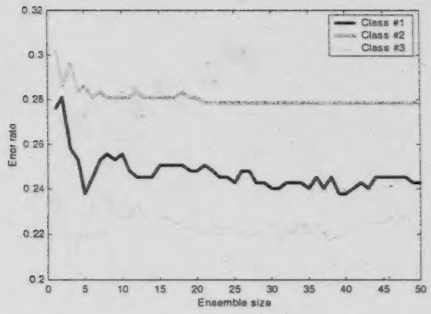
Figure 69: "Difficult" (to classify) data point with the superimposed data points coming from the testing set.

The results reported below quantify the performance of the ensemble of FALNs for different classification problems.

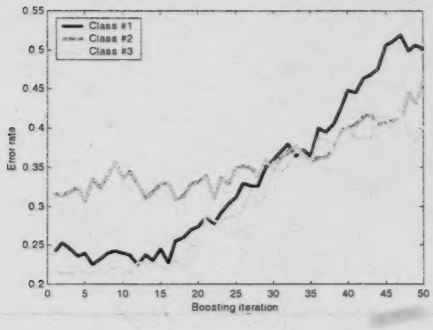
Classification problem #1: 3 classes: class_1: (number of changes ≤1 (with 114 instances); class_2: number of changes in (2,5) (106 instances), class_3: number of changes ≥5 (170 instances)

Classifier	Training set			Testing set		
	Class 1	Class 2	Class 3	Class 1	Class 2	Class 3
LDA	70.9±2.9	54.7±0.1	63.1±1.7	68.3±6.2	49.9±5.8	62.0±5.1
K-NN(5)	82.8±0.9	78.1±1.6	85.1±1.2	77.7±5.2	68.1±4.3	77.2±4.3
FALN(2,1)	81.3±1.7	74.1±0.6	81.6±2.0	75.4±3.3	71.7±1.4	80.0±4.3
FALN(3,2)	81.0±0.9	76.3±1.9	84.2±0.8	76.0±3.7	70.9±1.6	77.7±4.4
FALN(4,3)	81.7±1.7	75.5±2.2	86.1±0.2	76.5±4.5	68.9±4.9	77.2±3.1
FALN(6,2)	79.1±2.6	73.8±0.2	81.2±2.2	75.4±4.3	72.2±0.9	79.0±3.7

Table 26: Classification problem MIS #1: classification rates (along with their standard deviation)



(a)



(b)

Figure 70: Classification rates for the ensemble of FALNs; FALN(3,2) for the MIS classification problem #1 treated as a function of ensemble size: bagging (a) and boosting (b)

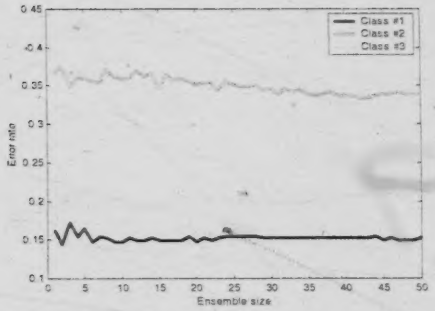
Classifier	Correct classification rate		
	Class 1	Class 2	Class 3
FALN (2,1)	75.4±3.3	71.7±1.4	80.0±4.3
Bagging (2,1)	77.5±4.9 (9)	69.6±3.9 (8)	81.2±3.6 (44)
Boosting (2,1)	78.0±4.4 (3)	65.6±2.4 (1)	78.5±2.7 (1)
FALN (3,2)	76.0±3.7	70.9±1.6	77.7±4.4
Bagging (3,2)	76.2±3.3 (5)	72.2±0.9 (21)	79.5±2.6 (3)
Boosting (3,2)	77.5±3.9 (12)	69.4±3.3 (5)	78.5±3.2 (2)

Table 27: Classification problem MIS #1: classification rates (± standard deviation) and their comparison with the ensemble methods

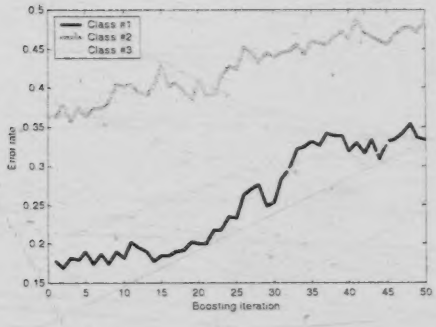
Classification problem #2: 3 classes: class_1: (number of changes =0 (51 instances); class_2: number of changes in (0,6) (190 instances), class_3: number of changes ≥6 (149 instances)

Classifier	Training set			Testing set		
	Class 1	Class 2	Class 3	Class 1	Class 2	Class 3
LDA	87.4±0.6	58.5±0.8	70.2±3.2	84.6±3.4	59.0±3.4	67.6±4.9
K-NN(9)	87.5±0.5	72.0±0.5	80.6±0.9	85.6±1.1	64.0±4.6	78.5±4.0
FALN(2,1)	88.1±1.1	68.6±1.4	77.7±1.5	86.1±0.9	63.0±2.7	76.2±4.4
FALN(3,2)	88.6±0.7	68.3±0.7	78.5±2.3	86.1±1.8	64.1±5.0	77.0±4.5
FALN(4,3)	87.7±0.2	67.4±2.4	79.5±3.6	86.3±0.6	61.8±4.8	78.0±4.6
FALN(6,2)	87.8±0.2	63.5±9.3	80.4±1.7	86.3±0.6	58.5±4.0	78.2±2.4

Table 28: Classification problem MIS #2: classification rates (along with their standard deviation)



(a)



(b)

Figure 71: Classification rates for the ensemble of FALNs; FALN(2,1) for the MIS classification problem #2 treated as a function of ensemble size: bagging (a) and boosting (b)

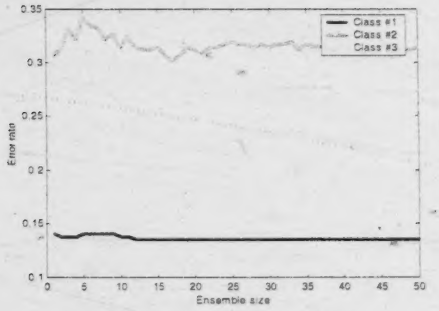
Classifier	Correct classification rate		
	Class 1	Class 2	Class 3
FALN (2,1)	86.1±0.9	63.0±2.7	76.2±4.4
Bagging (2,1)	85.6±1.9 (2)	66.6±2.6 (42)	80.8±4.7 (12)
Boosting (2,1)	83.0±2.5 (2)	64.3±3.7 (3)	79.2±4.0 (3)
FALN (3,2)	86.1±1.8	64.1±5.0	77.0±4.5
Bagging (3,2)	86.1±0.0 (5)	66.3±3.3 (29)	78.0±4.2 (5)
Boosting (3,2)	85.8±0.6 (1)	65.8±3.8 (2)	78.0±4.2 (6)

Table 29: Classification problem MIS #2: classification rates (± standard deviation) and their comparison with the ensemble methods

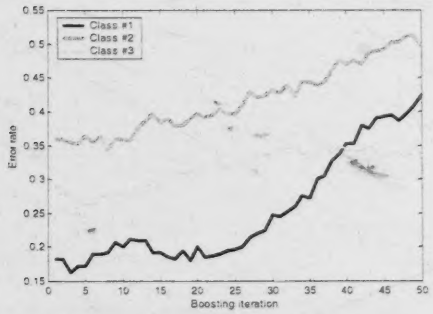
Classification problem #3: 3 classes: class_1: (number of changes =0 (51 instances); class_2: number of changes in (0,4) (141 instances), class_3: number of changes ≥4 (198 instances)

Classifier	Training set			Testing set		
	Class 1	Class 2	Class 3	Class 1	Class 2	Class 3
LDA	87.1±1.1	59.2±3.8	60.7±5.4	84.7±2.7	54.4±3.5	58.7±2.2
K-NN(3)	90.0±0.6	82.1±1.0	87.2±1.1	84.5±2.1	66.2±3.6	75.0±6.4
FALN(2,1)	87.7±0.4	70.8±5.5	74.8±2.9	86.3±0.0	64.8±2.4	72.8±3.2
FALN(3,2)	87.6±0.2	71.0±4.2	75.4±3.5	86.5±0.6	66.3±4.2	73.5±5.2
FALN(4,3)	87.6±0.2	70.4±3.4	76.6±4.2	86.3±0.0	68.0±2.7	71.0±4.2
FALN(6,2)	87.7±0.2	67.0±4.7	75.7±3.4	86.5±0.6	64.8±1.4	73.3±5.2

Table 30: Classification problem MIS #3: classification rates (along with their standard deviation)



(a)



(b)

Figure 72: Classification rates for the ensemble of FALNs; FALN(3,2) for the MIS classification problem #3 treated as a function of ensemble size: bagging (a) and boosting (b)

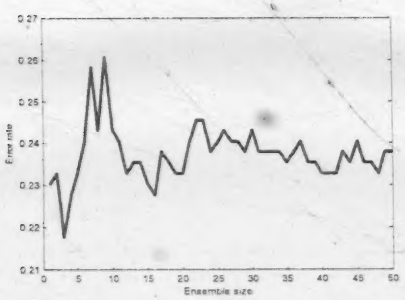
Classifier	Correct classification rate		
	Class 1	Class 2	Class 3
FALN (2,1)	86.3±0.0	64.8±2.4	72.8±3.2
Bagging (2,1)	85.5±1.7 (2)	66.5±4.7 (4)	76.3±3.2 (4)
Boosting (2,1)	83.8±3.2 (1)	66.2±4.1 (2)	74.5±3.4 (4)
FALN (3,2)	86.5±0.6	66.3±4.2	73.5±5.2
Bagging (3,2)	86.5±0.6 (12)	69.8±8.4 (17)	74.3±5.3 (17)
Boosting (3,2)	83.7±3.6 (3)	65.5±5.0 (8)	73.8±5.0 (5)

Table 31: Classification problem MIS #3: classification rates (± standard deviation) and their comparison with the ensemble methods

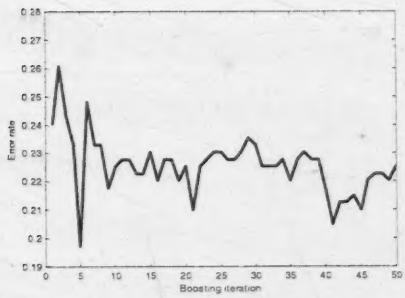
Classification problem #4: 2 classes: class_1: (number of changes ≤10 (308 instances);
class_2: number of changes >10 (82 instances)

Classifier	Training set	Testing set
LDA	76.2±2.5	71.4±6.1
K-NN(5)	84.5±1.2	79.0±5.8
FALN (2,1)	81.1±1.5	78.7±1.1
FALN (3,2)	81.9±3.3	79.0±3.4
FALN (4,3)	81.0±2.9	77.7±2.5
FALN (6,2)	80.3±0.7	77.5±3.0

Table 32: Classification problem MIS #4: classification rates (along with their standard deviation)



(a)



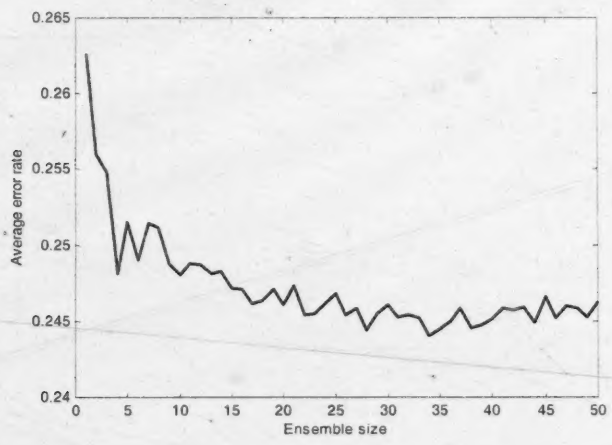
(b)

Figure 73: Classification rates for the ensemble of FALNs; FALN(3,2) for the MIS classification problem #4 treated as a function of ensemble size: bagging (a) and boosting (b)

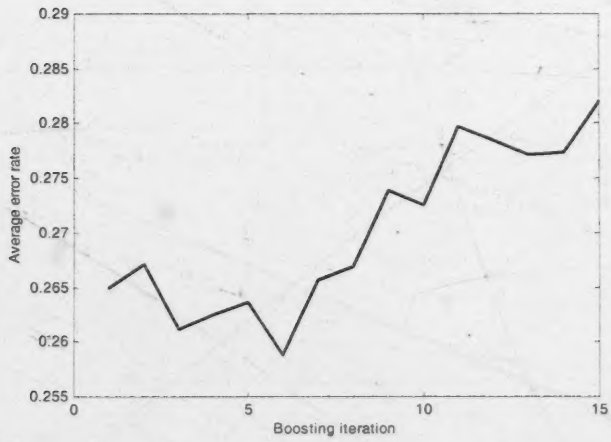
Classifier	Correct classification rate
FALN (2,1)	78.7±1.9
Bagging (2,1)	79.2±2.9 (2)
Boosting (2,1)	78.0±6.1 (1)
FALN (3,2)	78.0±3.2
Bagging (3,2)	78.2±4.6 (3)
Boosting (3,2)	80.2±4.5 (5)
FALN (4,3)	77.7±3.3
Bagging (4,3)	77.7±4.0 (2)
Boosting (4,3)	77.7±2.2 (8)

Table 33: Classification problem MIS #4: classification rates (± standard deviation) and their comparison with the ensemble methods

Figure 74 summarizes the results by visualizing averaged error rates for the FALNS in the experiments. Noticeably, increasing the size of the network ensemble becomes effective only up to some limit (say up to 10 networks) and afterwards the improvement is not substantial and the design of the larger ensemble of the networks is difficult to justify.



(a)



(b)

Figure 74: Averaged error rates results for all networks and all MIS classification problems: bagging (a) and boosting (b)

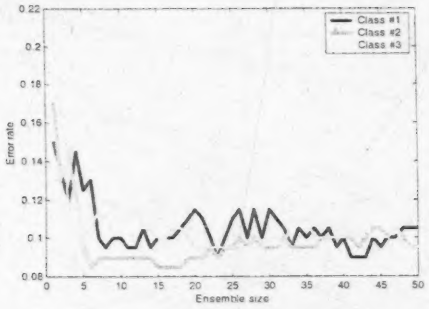
The experiments lead us to several interesting conclusions. Both bagging and boosting produce some

improvement over a single FALN. From analyzing the above experiments, in 70% of the experiments bagging was more accurate than a single network, achieving an average of about 2% improvement in accuracy, and a maximum improvement of nearly 5%. Boosting was more accurate than a single network in about 50% of the cases with an average improvement of 1.5% in accuracy, with a maximum improvement of 3%. The ensemble of 20–25 networks in the bagging approach led to some improvement. Likewise, in the case of boosting, an ensemble of around 5 networks enhanced the results.

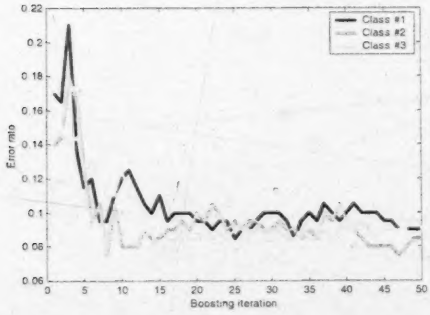
6.3.2 High dimensional biomedical data

Following are results obtained using ensembles of FALNs with biomedical data (described in Chapter 3): IR spectra of synovial fluid, MR spectra of brain neoplasms and MR spectra of a biological fluid.

Classification of IR spectra of synovial fluid



(a)

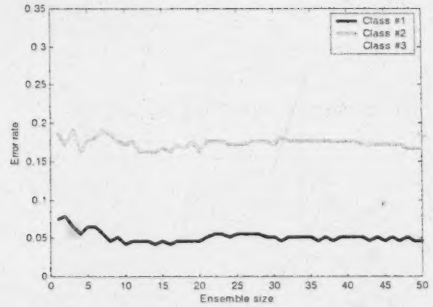


(b)

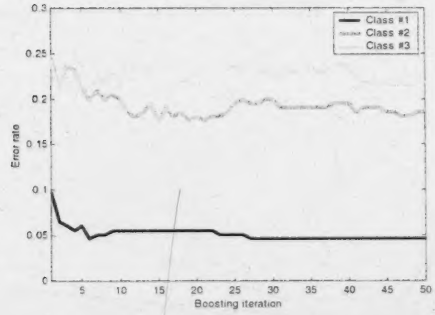
Figure 75: Classification rates for the ensemble of FALNs; FALN(3,1) for IR spectra of synovial fluid treated as a function of ensemble size: bagging (a) and boosting (b)

Classifier	Correct classification rate		
	Class 1	Class 2	Class 3
FALN (2,1)	90.0±5.3	91.0±7.8	91.5±2.8
Bagging (2,1)	89.5±5.4 (3)	92.0±5.4 (2)	91.5±2.2 (33)
Boosting (2,1)	92.5±5.0 (41)	93.0±3.3 (16)	92.0±5.4 (46)
FALN (3,1)	89.5±6.5	90.5±6.0	91.0±1.4
Bagging (3,1)	91.0±2.9 (41)	91.5±5.5 (6)	90.5±2.1 (20)
Boosting (3,1)	91.5±3.8 (33)	92.5±3.1 (8)	92.5±2.5 (15)

Table 34: IR spectra of synovial fluid: classification rates (± standard deviation) and their comparison with the ensemble methods



(a)



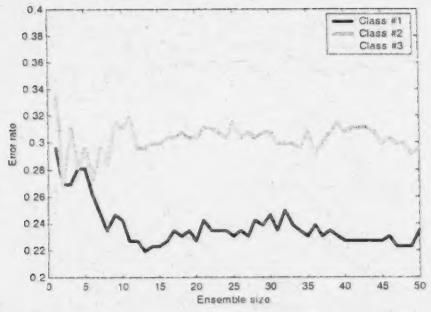
(b)

Figure 76: Classification rates for the ensemble of FALNs; FALN(3,1) for MR spectra of brain neoplasms treated as a function of ensemble size: bagging (a) and boosting (b)

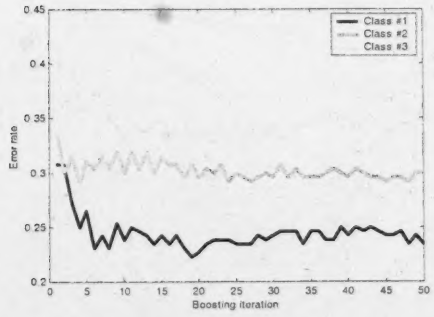
Classifier	Correct classification rate		
	Class 1	Class 2	Class 3
FALN (3,1)	94.9±1.9	76.2±3.4	73.5±7.3
Bagging (3,1)	95.8±3.0 (10)	83.7±7.7 (4)	79.5±7.2 (8)
Boosting (3,1)	95.3±3.7 (6)	82.3±8.6 (15)	80.0±4.5 (4)
FALN (4,1)	93.5±1.9	77.7±5.1	78.1±6.7
Bagging (4,1)	95.8±1.9 (11)	84.6±7.2 (8)	79.5±5.3 (2)
Boosting (4,1)	96.7±2.6 (44)	82.8±6.7 (12)	79.1±2.9 (2)

Table 35: MR spectra of brain neoplasms: classification rates (± standard deviation) and their comparison with the ensemble methods

Classification of MR spectra of a biological fluid



(a)



(b)

Figure 77: Classification rates for the ensemble of FALNs; FALN(2,1) for MR spectra of a biological fluid treated as a function of ensemble size: bagging (a) and boosting (b)

	Correct classification rate		
	Class 1	Class 2	Class 3
FALN (2,1)	74.2±7.8	71.5±6.4	61.2±6.8
Bagging (2,1)	78.1±8.7 (13)	73.1±4.1 (2)	66.1±6.1 (6)
Boosting (2,1)	78.1±9.7 (19)	70.8±3.4 (24)	67.7±5.7 (18)
FALN (2,2)	73.8±10	68.5±4.2	60.8±5.2
Bagging (2,2)	78.1±7.6 (28)	71.9±3.5 (42)	66.2±4.6 (2)
Boosting (2,2)	78.5±11 (43)	72.3±7.0 (2)	65.8±3.7 (4)

Table 36: A biological fluid: classification rates (± standard deviation) and their comparison with the ensemble methods

Summarizing the classification results for the three classification problems: again, both bagging and

boosting produce some improvement over a single FALN. In over 80% of the experiments bagging was more accurate than a single network, achieving an average of about 3.5% improvement in accuracy, and a maximum improvement of 7.5%. Boosting was more accurate than a single network in more than 90% of the cases with an average improvement of 1.5% in accuracy, with a maximum improvement of 3.3%. Similarly to the MIS data set, the ensemble of 20–25 networks in the bagging approach usually led to some improvement; however, in the case of boosting, the number of networks that brings the most significant improvement is somewhat inconclusive.

6.4 Summary

A first attempt to combine FALNs in ensembles, using bagging and boosting methods is described. Experiments were carried out on two types of data sets of entirely different character. The chapter includes a detailed analysis of the performance achieved with ensembles and a comparison with a single FALN for each case. It was shown that very often a significant improvement in classification accuracy could be obtained when using FALNs in an ensemble, therefore justifying the use of this method. It is also worth noting that the results described in this chapter are quite consistent with other experimental studies involving ensembles of neural networks (Opitz and Maclin, 1999).

7. Conclusions and recommendations for future work

7.1 Conclusions

This study explores the use of computational intelligence methods for pattern recognition, specifically high dimensional pattern classification. The two main building blocks of a pattern classification system are a feature extractor and the classifier (the actual assignment of a class label to a feature vector). Intentionally, these two issues are dealt with separately: Chapter 4 investigates a feature extraction method, whereas Chapters 5 and 6 deal with classifier design.

The purpose of keeping these two topics here independent from one another is to enable focusing separately on each of them. Generally, in pattern classification, there is no such thing as the "best" feature extractor and classifier. In approaching a specific classification problem, the issue of combining the most appropriate feature extractor with the most suitable classifier is of vital importance for achieving the optimal classification system. Each classification problem requires careful attention in selecting the appropriate methods, integrating between them and optimizing their parameters.

Three major parts make up this study. The first part explores a piecewise polynomial curve approximation method, based on GA optimization. It is demonstrated specifically for the use of piecewise linear representation of biomedical spectral data and it is shown to have comparable and at times better performance with respect to another well known algorithm (bottom-up segmentation). Moreover, this proposed method is conveniently extended for the purpose of optimizing a set of breaking points that is common to a set of curves (more generally, feature vectors), rather than a single one. This expansion can be seen as a basis for a feature extraction method.

The second part is concerned with a class of fuzzy adaptive logic networks in which the geometry of the feature space is captured through a collection of linear classifiers (perceptrons) while the aggregation of the results is realized in the logic framework. The logic processors combine operations that come with clear semantics of *or*- and *and*-like summarization. Several interpretation issues of these networks were discussed along with illustrative examples, and the effectiveness of fuzzy adaptive logic networks as a system for classifying complex biomedical spectra is demonstrated.

Lastly, the third part includes a detailed presentation and analysis of ensembles of FALNs (bagging and boosting). While ensemble methods have been in use during the last decade or so, they were investigated mostly in connection with neural networks and classification trees. Demonstrated here is a first attempt to combine FALNs in an ensemble, and this was experimented using software engineering data and biomedical data. The comprehensive experimental results are fully reported and carefully quantified.

7.2 Future work

The research involving GA-optimized piecewise curve approximation can be expanded in interesting ways. This study dealt with piecewise polynomial representation using primarily linear segments (and some work has been done with order 2 polynomials also). In all cases it has been assumed that the polynomials for all sections of the curves are of uniform order. Since different sections of a curve may be of different nature, it could be advantageous, in some cases, to create a representation in which, for example, some of the segments would be represented by a linear function, while others – by higher order polynomials. Thus creating a *piecewise mixed-order polynomial representation*. The set of parameters to optimize in this case would then be expanded to include the polynomial order for each section, in addition to the breaking point locations. In addition to that, the fitness function would have to be modified to take into account the total number of parameters required to represent a curve along with the achieved accuracy since, especially in classification problems, it can be significantly beneficial to have a representation that is as compact as possible.

Another topic, enfolded in the piecewise curve approximation method, that can be investigated concerns the issue of constructing a representation that would satisfy the *continuity* requirement. The method presented in this thesis calculates an optimal piecewise representation in terms of accuracy, however, in some cases there can be, additionally, a continuity constraint. An interesting avenue to pursue could be the idea of *fuzzy segmentation points* where a membership function, say in the form of a Gaussian, is placed around the breaking point locations. The approximation value in these regions could then be expressed as a combination of the two end-points values of the respective segments. It should be instructive to investigate the possibility of achieving a more accurate representation with this expansion, since it is not engaged in an increase in the number of parameters used for the representation.

As for future studies with regard to FALNs, one can consider the proposed topology as a general framework of logic-oriented neuro-computing consisting of a combination of hyper-planes and radial basis functions (RBFs), which focus on *local* processing. With the existing diversity of the structure of patterns, one may envision a hybrid structure that can be combined with the logic-based processing module and form

a highly transparent topology. An example of the network of this nature (being in fact a generalization of the FALN structure) is depicted in Figure 78. Note that there is a diversity of global processing units and local receptive fields that form a processing interface with the logic module aggregating all these partial results.

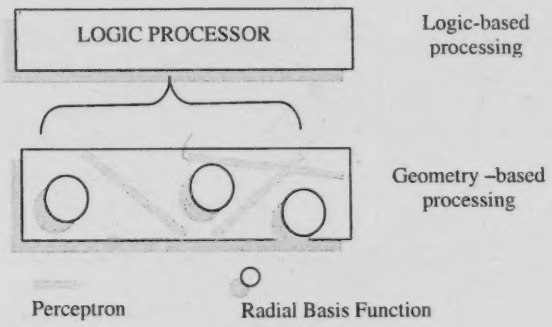


Figure 78: A heterogeneous neural structure with the logic layer interacting with the numeric interface constructed by perceptrons and RBFs

Evidently, in the design of the heterogeneous structure one envisions a facet of important structural optimization where a number of questions arise: what type of RBFs to select, what mix of local and global processing units to select, etc. Definitely, this type of learning is beyond the scope of the parametric gradient-based learning discussed in Section 4. In this setting, an interesting topic could be genetic optimization (Russo, 2000) that is aimed at the structural optimization (that is a structuralization of the receptive fields and perceptrons). This phase is followed by a detailed parametric optimization of the blueprint of the network.

8. References

Armstrong, W. W., Chu, C., Thomas, M. M. (1995) "Using adaptive logic networks to predict machine failure," in *Proc. World Congress on Neural Networks*, Washington, DC, 2: 80-83

Bandyopadhyay, S., Pal, S. K. (1999) "Relation between VGA-classifier and MLP: determination of network architecture," *Fundamenta Informaticqe*, 37(1-2): 177-199

Basu, M., Ho, T. K. (1999) "The Learning Behavior of Single Neuron Classifiers on Linearly Separable or Nonseparable Input," in *Proc. IEEE Intl. Joint Conf. on Neural Networks (IJCNN'99)*, Washington, DC

Bauer, E., Kohavi, R. (1999) "An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants," *Machine Learning*, 36: 105-142

Beasley, D., Bull, D. R., Martin, R. R. (1993) "An overview of genetic algorithms: Part 1, Fundamentals," *University Computing*, 15(2): 58-69

Bellman, R. (1961) Adaptive Control Processes: A Guided Tour, Princeton University Press, Princeton

Bennet, K.P. (1992) "Decision tree construction via linear programming," in *Proceedings of the 4th Midwest Artificial Intelligence and Cognitive Society Conference*, Hume, G., Evans, M. (Eds.), Utica IL, pp. 97-101

Bezdek, J.C. (1994) "What is Computational Intelligence?" in Computational Intelligence Imitating Life, Zurada, J. M., Marks R. J. II, Robinson C. J. (Eds.), IEEE Press, pp. 1-12

Bezdek, J.C. (1992) "On the Relationship between Neural Networks, Pattern Recognition and Intelligence," *International Journal of Approximate Reasoning*, 6: 85-107

Bioch, J. C., Carsouw, R., Potharst, R. (1995) "On the use of simple classifiers for the initialisation of one-hidden-layer neural nets," Department of Computer, Erasmus University Rotterdam, P.O Box 1738, 3000 DR Rotterdam, The Netherlands. Technical report eur-cs-95-08

Bishop, M. (1995) Neural Networks for Pattern Recognition, Oxford University Press, Oxford

Bradley, P (1995) Multisurface Method Tree with MATLAB, http://www.cs.wisc.edu/~olvi/uwmp/msmt.html

Breiman, L. (1996a) "Bagging predictors," *Machine Learning*, 24(2): 123-140

Breiman, L. (1996b) "Bias, Variance and Arcing Classifiers," Technical Report 460, Statistics Department, University of California, Berkeley, *April*

Chambers, L. (2000) The Practical Handbook of Genetic Algorithms, 2nd Edition, CRC Press

Chi, Z., Yan, H., Pham, T. (1996) Fuzzy Algorithms: With Applications to Image Processing and Pattern Recognition, World Scientific

Cogger, K. O. (1997) "An Introduction to Adaptive Logic Networks With an Application to Audit Risk Management," *American Accounting Association Meeting*, Dallas, *August*

Cover, T., Hart, P. (1967) "Nearest neighbor pattern classification," *IEEE Trans. Information Theory*, 13: 21-27

Dasarathy, B. V. (1991) Nearest-Neighbor Classification Techniques, IEEE Computer Society Press, Los Alomitos, CA

Dietterich, T. G. (2000) "An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization," *Machine Learning*, 40(2): 139-157

Dipersio, D. A., Barr, R. C. (1985) "Evaluation of the fan method of adaptive sampling on human electrocardiograms," *Medical & Biological Engineering & Computing*, pp. 401–410, *September*

Douglas, D., Peuker, T. (1973) "Algorithms for the reduction of the number of points required to represent a digitised line or its caricature," *The Canadian Cartographer*, 10: 112-122

Duch, W., Adamczak R. (1998) "Statistical methods for construction of neural networks," in *Proc. International Conference on Neural Information Processing, ICONIP'98*, Kitakyushu, Japan, 2: 629-642, *October*

Duda, R. O., Hart, P. E. (1973) Pattern Classification and Scene Analysis, John Wiley & Sons, NY

Duda, R. O., Hart, P. E., Stork D. G. (2001) Pattern Classification, 2nd Edition, John Wiley & Sons, NY

Fisher, R. A. (1936) "The use of multiple measurements in taxonomic problems," *Annals of Eugenics*, 7: 179-188

Fukunaga, K. (1990) Introduction to Statistical Pattern Recognition, 2nd Edition, Academic Press, San Diego, California

Freund, Y., Schapire, R.E. (1996) "Experiments with a New Boosting Algorithm," *Proceedings of the International Conference on Machine Learning (ICML-96)*, pp. 148-156

Gabrys, B., Bargiela, A. (2000) "General fuzzy Min-Max neural network for clustering and classification," *IEEE Trans. on Neural Networks*, 11(3): 769-783

Gacek, A., Pedrycz, W. (2003) "A Genetic Segmentation of ECG Signals," *IEEE Transactions on Biomedical Engineering*, to appear

Gaudet, V. C. (1996) "Genetic Programming of Logic-Based Neural Networks," in Genetic Algorithms for Pattern Recognition, Pal, S. K., Wang, P. P. (Eds.), CRC Press, Boca Raton, pp. 213-226

Goldberg, D. E. (1989) Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, Reading, MA

Hagan, M. T., Menhaj, M. B. (1994) "Training Feedforward Networks with the Marquardt Algorithm," *IEEE Trans. on Neural Networks*, 5(6): 989-993, *November*

Hansen, L. K., Salamon (1990) P. "Neural network ensembles," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12: 993-1001

Hasenjager, M., Ritter, H. (1999) "Perceptron Learning Revisited: The Sonar Targets Problem," *Neural Processing Letters*, 10: 1-8

Hassoun, M. H. (1995) Fundamentals of Artificial Neural Networks, MIT Press, Cambridge, MA

Haykin, S. (1994) Neural Networks: a Comprehensive Foundation, Macmillan College Publishing Company

Heckbert, P. S., Garland, M. (1997) "Survey of polygonal surface simplification algorithms," *Multiresolution Surface Modeling Course Notes, SIGGRAPH'97*

Herrera, F., Lozano, M. J., Verdegay, L. (1998) "Tackling real-coded genetic algorithms: Operators and tools for behavioral analysis," *Artificial Intelligence Review*, 12: 265-319

Hershberger, J., Snoeyink, J. (1992) "Speeding Up the Douglas-Peucker Line-Simplification Algorithm," in *Proc. 5th Symp on Data Handling*, pp. 134-143

Holland, J. H. (1975) Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, MI

Imai, H., Iri, M. (1986) "An optimal Algorithm for Approximating a Piecewise Linear Function," *Journal of Information Processing*, 9(3): 59-62

Ishibuchi, H., Murata, T., Tanaka, H. (1996) "Construction of Fuzzy Classification Systems with Linguistics If-Then Rules Using Genetic Algorithms," in Genetic Algorithms for Pattern Recognition, Pal, S. K., Wang, P. P. (Eds.), CRC Press, Boca Raton, pp. 227-252

Ishibuchi, H., Nozaki, K., Yamamoto, N., Tanaka, H. (1995) "Selecting fuzzy if-then rules for classification problems using genetic algorithms," *IEEE Transactions on Fuzzy Systems*, 3(3): 260-270

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., Hinton, G. E. (1991) "Adaptive mixtures of local experts," *Neural Computation*, 3: 79-87

Jang, J. -S. R., Sun, C. -T., Mizutani, E. (1997) Neuro-Fuzzy and Soft Computing: A Computational Approach to Machine Intelligence, Prentice Hall, Englewood Cliffs, NJ

Jin, Y., Ma, S. (2000) "A Neural-Network Dimension Reduction Method for Large-Set Pattern Classification," in Tan, T., Shi, Y., Gao, W. (Eds.), *ICMI 2000, LNCS 1948*, pp. 426-433

Jolliffe, I. T. (1986) Principal Component Analysis, Springer-Verlag, NY

Jordan, M. I., Jacobs, R. A. (1994) "Hierarchical mixture of experts and the EM algorithm," *Neural Computation*, 6: 181-214

Keogh, E. (2003) http://www.cs.ucr.edu/~eamonn/TSDMA/time_series_toolbox/

Keogh, E., Chu, S., Hart, D., Pazzani, M. (2001) "An Online Algorithm for Segmenting Time Series," in *Proceedings of IEEE International Conference on Data Mining*, pp 289-296

Lind, R. K., Vairavan, K. (1989) "An Experimental Investigation of Software Metrics and Their Relationship to Software Development Effort," *IEEE Transactions on Software Engineering* 15(5): 649-653

Maclin, R., Opitz, D. (1997) "An Empirical Evaluation of Bagging and Boosting," in *Proceedings of the fourteenth International Conference on Artificial Intelligence*, pp. 546-551, Cambridge, MA. AAAI Press/MIT Press

Mangasarian, O. L. (1993) "Mathematical Programming in neural networks," *ORSA Journal on Computing*, 5(4): 349-360

Michalewicz, Z. (1992) Genetic Algorithms + Data Structures = Evolutionary Programs, Springer-Verlag, Berlin, Germany

Mitra, S., Hayashi, Y. (2000) "Neuro–Fuzzy Rule Generation: Survey in Soft Computing Framework," *IEEE Trans. on Neural Networks*, 11(3): 748-768

Munson, J. C., Khoshgoftaar, T. M. (1996) "Software metrics for reliability assessment," in: Handbook of Software Reliability and System Reliability, McGraw-Hill Inc., Hightstown, NJ

Murphy, P. M., Aha, D. W. (1994) UCI repository of Machine Learning Databases, University of California, Department of Information and Computer Science, Irvine, CA

Nygaard, R., Husøy, J. H., Haugland, D., Aase, S. O. (1999) "Signal compression by piecewise linear non-interpolating approximation," in *Proc. of International Conference on Acoustics, Speech, and Signal Proc. (ICASSP)*, Phoenix, Arizona, USA, pp. 1273-1276, *March*

Opitz, D., Maclin, R. (1999) "Popular Ensemble Methods: An Empirical Study," *Journal of Artificial Intelligence Research*, 11: 169-198

Pedrycz, W. (1991) "Neurocomputations in relational systems," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 13: 289-296

Pedrycz, W. (1997) Computational Intelligence: An Introduction, CRC Press, Boca Raton, FL

Pedrycz, W. (1998) "Conditional fuzzy clustering in the design of radial basis function neural networks," *IEEE Transactions on Neural Networks*, 9(4): 601- 612

Pedrycz, W., Bargiela, A. (2002) "Granular clustering: a granular signature of data," *IEEE Trans. on Systems, Man, and Cybernetics –B*, 32(2): 212-224

Pedrycz, W., Breuer, A., Pizzi, J. N. (2003) "Fuzzy adaptive logic networks: at the junction of geometry and logic," *Pattern Recognition* (submitted)

Pedrycz, W., Gomide, F. (1998) An Introduction to Fuzzy Sets, Cambridge, MIT Press, Cambridge, MA

Pedrycz, W., Rocha, A. (1993) "Knowledge-based neural networks," *IEEE Trans. on Fuzzy Systems*, 1: 254-266

Pedrycz, W., Vasilakos, A. V. (1999) "Linguistic models and linguistic modelling," *IEEE Trans. on Systems Man and Cybernetics*, 29(6): 745-757

Pittman, J., Murthy, C. A. (2000) "Fitting Optimal Piecewise Linear Functions Using Genetic Algorithms," *IEEE Pattern Analysis and Machine Intelligence*, 22(7): 701-718

Pittman, J., Murthy, C. A. (1997) "Optimal Line Fitting Using Genetic Algorithms," Technical Report 99-02, University Park, The Pennsylvania State University, Department of Statistics, *July*

Press, W. H., Flannery, B. P., Teukolsky, S. A., Vetterling, W. T. (1992) Numerical Recipes in C: The Art of Scientific Computing, 2nd edition, Cambridge University Press, Cambridge

Quinlan, J. (1996) "Bagging, Boosting, and C4.5," in *Thirteenth National Conference on Artificial Intelligence*, (Cambridge), AAAI Press/MIT Press, pp. 163-175

Ramurthi, V., Ghosh, J. (1996) "Structural adaptation in mixture of experts," *Proc. of ICPR96*, pp. 704-708

Ripley, B. D. (1996) Pattern Recognition and Neural Networks, Cambridge University Press

Rumelhart, D. E., Hinton, G. E., Williams, R. J. (1986) "Learning Internal Representations by Error Propagation," in Parallel Distributed Processing, vol. 1, The MIT Press

Russo, M. (2000) "Genetic fuzzy learning," *IEEE Transactions on Evolutionary Computation*, 4(3): 259-273

Saupe, D. (1997) "Optimal piecewise linear image coding," in *Proc. SPIE, Conf. Visual Communications and Image Processing*, 1997, 3309: 747–760

Schaffer, D., Whitley, D., Eshelman, L. (1992) "Combinations of Genetic Algorithms and Neural Networks: A survey of the state of the art", in *Proceedings of the International Workshop on Combinations of Genetic Algorithms and Neural Networks*, Whitley, D., Schaffer, D. (Eds.), IEEE Computer Society Press, Los Alamitos, CA, pp. 1-37

Schapire, R. E. (2002) "The Boosting Approach to Machine Learning: an Overview", in *MSRI Workshop on Nonlinear Estimation and Classification*

Schapire, R. E. (1990) "The strength of weak learnability", *Machine Learning*, 5(2): 197– 227

Schmidhuber, J., Hochreiter, S. (1996) "Guessing can outperform many long time lag algorithms," Technical Note *IDSIA-19-96, May*

von Schmidt, B. , Klawonn, F. (2001) "Construction of a Multilayer Perceptron for a Piecewise Linearly Separable Classification Problem," *IFSA / NAFIPS 2001*, paper ID 231

Simpson, P. K. (1992) "Fuzzy Min-Max neural networks – Part 1: Classification," *IEEE Trans. on Neural Networks*, 3(5): 776-86, *September*

Simpson, P. K. (1993) "Fuzzy Min-Max neural networks – Part 2: Clustering," *IEEE Trans. on Neural Networks*, 4(1): 32-45, *February*

Sudkamp, T. A., Hammell II, R. J. (1998) "Granularity and specificity in fuzzy function approximation," in *Proc. NAFIPS-98*, pp. 105-109

White, E. R. (1985) "Assessment of line-generalization algorithms using characteristic points," *American Cartographer*, 12(1): 17-27

Zadeh, L. A (1979) "Fuzzy sets and information granularity," in: Gupta, M. M., Ragade, R. K., Yager, R. R. (Eds.), Advances in Fuzzy Set Theory and Applications, North Holland, Amsterdam, pp. 3-18

Zadeh, L. A. (1996) "Fuzzy logic = Computing with words," *IEEE Trans. on Fuzzy Systems*, 4(2): 103-111

Zadeh, L. A., (1997) "Toward a theory of fuzzy information granulation and its centrality in human reasoning and fuzzy logic," *Fuzzy Sets and Systems*, 90: 111-117